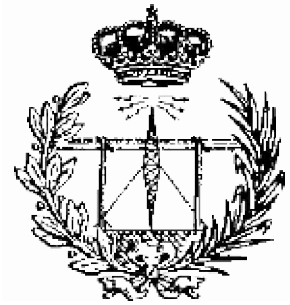
ESCUELA SUPERIOR DE INGENIEROS
UNIVERSIDAD DE SEVILLA
Ingeniería Superior de Telecomunicación

Proyecto Fin de Carrera



Simulación de convertidores pipelined

Autor del Proyecto
Daniel Falcón Medina

Tutor del Proyecto
Fernando Muñoz Chavero

Departamento de Ingeniería Electrónica, Grupo de Tecnología Electrónica.
Universidad de Sevilla

Sevilla, Julio 2005

A mis padres, mi hermano y mi novia

A mi tutor, Fernando Muñoz Chavero,
por su ayuda y dedicación

A todos los que me han apoyado

Índice

Capítulo 1. Introducción 1

1.1 Motivación y contexto.....1
1.2 Objetivos del proyecto.....2
1.3 Organización del proyecto.....3

Capítulo 2. Conversión Analógico/Digital 5

2.1 Introducción…….....5
2.2 Conceptos básicos.....6
2.2.1 Filtro analógico antisolapamiento.....7
2.2.2 Muestreo…….....7
2.2.2.1 Teorema de muestreo.....8
2.2.3 Cuantización…….....8
2.2.4 Codificación…….....9
2.2.5 Conversión de la tasa de muestreo.....9
2.3 Convertidor A/D.....10
2.3.1 Figuras de mérito…….....10
2.3.1.1 Relación señal a ruido.....10
2.3.1.2 Rango dinámico…….....10
2.3.1.3 Relación señal a ruido y distorsión.....11
2.3.1.4 Número efectivo de bits.....11

Capítulo 3. Convertidores A/D pipeline 12

3.1 El convertidor pipeline.....12
3.2 Principales no idealidades en un CA/D Pipeline.....14
3.2.1 Ganancia DC Finita del amplificador operacional.....15
3.2.2 Tamaño de las capacidades.....15
3.2.2.1 Ruido térmico…….....15
3.2.2.2 Desapareamiento entre capacidades.....16
3.2.3 Slew Rate y Ancho de Banda del amplificador operacional.....17
3.2.4 Offset en los DACs…….....19
2.3.1 Jitter….....19

Capítulo 4. Convertidores A/D Flash 20

4.1 El convertidor flash.....20
4.2 Principales no idealidades en un CA/D Flash.....22

Capítulo 5. Interfaces gráficas de usuario en Matlab 23

5.1 Introducción.....23
5.2 Diseño de la GUI.....24
5.2.1 Creación o apertura de la interfaz de usuario.....24
5.2.1.1 Creación de una nueva GUI.....25
5.2.1.1.1 GUI en blanco.....26
5.2.1.1.2 GUI con Uicontrols.....26
5.2.1.1.3 GUI con Ejes y Menú.....27
5.2.1.1.4 Modal Question.....28
5.2.1.2 Apertura de una GUI ya existente.....29
5.2.2 Edición de la interfaz de usuario.....30
5.3 Programación de la GUI.....32

Capítulo 6. Operaciones en MATLAB **33**

6.1 Introducción.....33
6.2 Script de la GUI: Funciones y script auxiliares.....34

Capítulo 7. Librerías y modelos **35**

7.1 Introducción.....35
7.2 Librerías.....35
7.2.1 Mylibrary.mdl.....36
7.2.2 Substages.mdl.....39
7.2.3 Digcorrect.mdl.....39
7.2.4 Parts6bits.mdl.....40
7.3 Modelos.....42

Capítulo 8. Resultados de las simulaciones en Matlab **45**

8.1 Introducción.....45
8.2 Pipeline.....46
8.3 Flash.....69

Capítulo 9. Conclusiones y futuras líneas de desarrollo **79**

9.1 Conclusiones.....79
9.2 Futuras líneas de desarrollo.....80

Bibliografía **81**

Apéndice 1. Manual de Usuario 82

A1.1 Introducción.....82
A1.2 Estructura de la GUI.....83
A1.2.1 Zona de Menús.....84
A1.2.1.1 Menú File.....84
A1.2.1.1.1 Menú File Submenú New.....85
A1.2.1.1.2 Menú File Submenú Open.....86
A1.2.1.2 Menú Environment.....87
A1.2.1.3 Menú Help.....89
A1.2.2 Zona de Non Idealities.....89
A1.2.3 Zona de Inputs.....90
A1.2.4 Zona de Messages.....92
A1.2.5 Zona de Graph Plotting.....93
A1.2.6 Zona de Results.....94
A1.2.7 Zona de Simulación.....95
A1.3 Ejemplo.....99

Apéndice 2. Código Matlab 104

Capítulo 1
Introducción

1.1 Motivación y Contexto

El carácter eminentemente analógico de nuestro entorno origina la necesidad de utilizar una interfaz para adecuar las distintas señales de información provenientes del mismo a los modernos sistemas de comunicaciones de voz y datos, donde se lleva a cabo digitalmente el procesado, cada vez más complejo, de estas señales analógicas. Esta interfaz anteriormente mencionada es el convertidor analógico-digital (CA/D).

Televisión de alta definición, multimedia, comunicaciones inalámbricas, receptores de radio, sistemas radar, detectores o modems son algunos ejemplos de aplicaciones potenciales de convertidores analógicos digitales.

Una aplicación que ha recibido una atención creciente en los últimos años es la transmisión de datos sobre el par trenzado de las líneas telefónicas sin interferir sobre el servicio telefónico clásico. Las líneas ADSL (Asymmetrical Digital Subscriber Line) proporcionan el ancho de banda necesario para aplicaciones tales como el acceso rápido a Internet o videoconferencia. Dado que la mayor parte del procesado de la señal se realiza en el dominio digital usando DSPs (Procesadores Digitales de Señal), se necesitarán convertidores, tanto analógico-digital (A/D) como digital-analógico (D/A), que actuarán de interfaces. La viabilidad del sistema ADSL dependerá en gran medida de la viabilidad de los convertidores, ya que las especificaciones típicas requieren altas resoluciones (entre 13 y 16 bits) para las señales de entrada.

La operación de conversión analógico a digital está sujeta a un compromiso entre la velocidad y la resolución en los CA/D. Esto es, una arquitectura válida para una velocidad alta de operación tendrá una resolución "baja" o no tan buena como su velocidad y viceversa para un mismo consumo de potencia.

Típicamente, los CAD de alta resolución, útiles por ejemplo en aplicaciones de video, se han basado, bien en convertidores de aproximaciones sucesivas con calibración, o bien en arquitecturas sobremuestreadas. Sin embargo, ninguna de estas arquitecturas es adecuada para alcanzar la velocidad prevista de conversión compatibles con los requisitos del ancho de banda de señal. Convertidores flash de dos pasos son bastante usados para resoluciones de conversión en el rango de los 8-12 bit, donde un diseño óptimo permite implementaciones de poca área de silicio y baja potencia disipada. De cualquier forma, por encima de esta resolución, los CA/D pipeline dan mejores resultados para minimizar al mismo tiempo potencia consumida y área.

La potencia disipada se minimiza escogiendo apropiadamente la resolución por etapa y optimizando la distribución de ruido entre varias etapas. Resoluciones de menos de un bit por etapa son ideales para convertidores pipeline de baja resolución, pero la resolución óptima por etapa aumenta a medida que la resolución total aumenta.

Las técnicas de calibración son muy importantes en el diseño de CA/D de alta resolución ya que la precisión de los componentes de fabricación es limitada, y raramente estable o bien caracterizada durante la vida útil de cualquier proceso. Por otro lado, es conveniente una calibración del convertidor en background, pues muchas aplicaciones tienen cortos periodos de inactividad y pueden existir posibles derivas con cambios de tensión y temperatura.

1.2 Objetivos del proyecto

Dado que los convertidores analógicos-digitales se encuentran en la interfaz analógica-digital, carecen de metodologías semiautomáticas para su diseño, como ocurre con los circuitos puramente digitales. Por lo tanto, se necesitan unas herramientas que ayuden a simular su diseño (comprobar el efecto de cambiar sus parámetros, como el tiempo de muestreo, el fondo de escala, etc., y de introducir en nuestros modelos algunas no idealidades, como la ganancia finita, el desapareamiento de capacidades, el jitter, etc.).

Este proyecto fin de carrera pretende satisfacer la necesidad de simuladores de comportamiento para convertidores A/D, mediante una interfaz amigable, versátil y de sencillo e intuitivo manejo. No se pretende, no obstante, desarrollar una herramienta definitiva sino establecer una base sobre la que poder construir nuevas ampliaciones y mejoras.

En particular, se fija como objetivo desarrollar un simulador de convertidores A/D bajo entorno MATLAB, programa para cálculo técnico y científico, con lenguaje de programación propio.

Este programa, para ciertas operaciones es muy rápido, cuando puede ejecutar sus funciones en código nativo con los tamaños más adecuados para aprovechar sus capacidades de vectorización.. En otras aplicaciones resulta bastante más lento que el código equivalente desarrollado en C/C++ o Fortran. Sin embargo esta herramienta ha sido seleccionada debido a que MATLAB es una magnífica herramienta de alto nivel para desarrollar aplicaciones técnicas, fácil de utilizar y que aumenta la productividad de los programadores a otros entornos de desarrollo. Asimismo, resulta interesante el hecho de que MATLAB dispone de un código básico y de varias librerías especializadas (toolboxes).

MATLAB permite desarrollar de una manera simple interfaces de usuario (ventana mediante la cual el programa se "comunica" con el usuario interactivamente mediante botones, menús desplegables, cuadros de texto, etc.), que permiten utilizar de manera muy simple programas realizados en el entorno de Windows.

1.3 Organización del proyecto

El proyecto ha sido organizado por capítulos del siguiente modo:

- Capítulo 1: Introducción: Motivación y contexto, objetivos y organización del proyecto.
- Capítulo 2: Conversión Analógico-Digital: Conceptos básicos de la conversión analógico-digital.
- Capítulo 3: Convertidores A/D Pipeline: En este capítulo se explican la estructura de un CA/D pipeline y las principales no idealidades que debemos considerar en su estudio.
- Capítulo 4: Convertidores A/D Flash: En este capítulo se explican la estructura de un CA/D flash y las principales no idealidades que debemos considerar en su estudio.
- Capítulo 5: Interfaces gráficas de usuario en Matlab: Este capítulo pretende ser una guía básica que describa el proceso de creación de una interfaz gráfica de usuario (GUI) en Matlab.
- Capítulo 6: Operaciones en Matlab: Cómo se estructura el programa implementado.
- Capítulo 7: Librerías y modelos: Librerías y modelos que se han dejado a disposición del usuario del programa.

- Capítulo 8: Resultados de las simulaciones en Matlab: En este capítulo se mostrarán los resultados obtenidos con la herramienta de simulación derivados del estudio de CA/D pipeline y flash.

- Capítulo 9: Conclusiones y futuras líneas de desarrollo: Conclusiones extraídas a la finalización del proyecto y propuesta de futuras mejoras y ampliaciones del mismo.

- Bibliografía: Compendio de la documentación usada para elaborar este proyecto.

- Apéndice 1: Manual de Usuario: Descripción general de la GUI del programa de simulación de convertidores A/D. Está pensado como guía básica, recogiendo de forma resumida todos los aspectos y posibilidades que tiene el usuario de la herramienta.

- Apéndice 2: Código Matlab: Listado completo del código comentado del programa.

Capítulo 2
Conversión Analógico/Digital

2.1 Introducción

Un Convertidor Analógico-Digital (CA/D) es un circuito electrónico que transforma una señal continua en el tiempo y en amplitud (señal analógica) en otra señal discreta en el tiempo y cuya amplitud está cuantificada y codificada, generalmente mediante un código binario (señal digital).

El CA/D es uno de los bloques esenciales que integran un sistema electrónico que utiliza procesamiento digital de señal.

Varias son las ventajas del procesamiento en el dominio digital de señales frente al mismo en el dominio analógico. Entre ellas podemos destacar:

- Robustez ante los ruidos e interferencias: Como las señales digitales se representan con un número discreto de valores eléctricos, resulta muy difícil para una perturbación contaminar una señal digital tanto como para llevarla a otro valor discreto. Esto no ocurre con una señal analógica, donde una pequeña perturbación puede variar el valor de dicha señal y alterar su posterior lectura y/o procesado.

- Diseño y desarrollo de circutería simple: Como en último término tenemos secuencias de valores discretos, el diseño es personalizable, de ahí el que cada vez más el diseño de circuitos digitales tenga más un carácter de *software* (o de programación) que de *hardware* (con elementos físicos). Esto se debe a que los circuitos digitales son más modulables que los analógicos, de manera que se puede diseñar un bloque digital y su funcionalidad no se altera sustancialmente cuando ese bloque se conecta a otros. Este no es el caso de la circutería analógica.

En la Figura 2.1 se ilustra un esquema general de un sistema de procesamiento digital de señales:

Figura 2.1.: Sistema de procesado digital de señales

La señal de salida del procesador digital puede necesitar nuevamente una conversión de formato, en este caso de digital a analógico, antes de ser entregada al receptor de la información (Figura 2.1); o, por el contrario, puede no requerir de ninguna conversión, preservando la señal el carácter digital.

2.2 Conceptos básicos

La conversión de una señal analógica a una señal digital incluye tres procesos fundamentales, que son:

- Prefiltrado antisolapamiento (o "antialiasing").
- Muestreo.
- Cuantización.

Los bloques asociados a estos procesos se muestran en la figura 2.2:

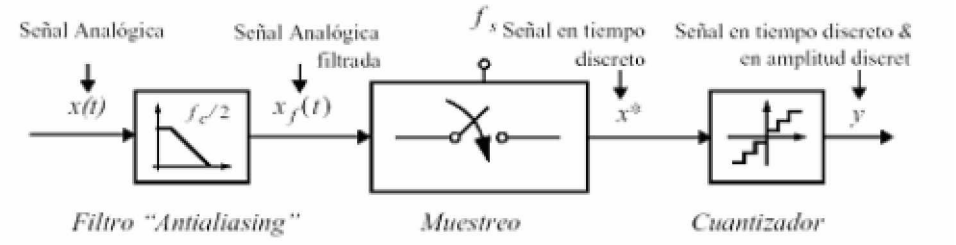


Figura 2.2.: Diagrama general de un convertidor A/D

Junto a estos tres procesos fundamentales veremos otros dos, que son:

- Codificación.
- Conversión de la tasa de muestreo.

2.2.1 Filtro analógico antisolapamiento

La señal analógica realmente nunca estará, estrictamente hablando, limitada en banda, por lo que filtramos la señal para que la entrada a nuestro muestreador sea una señal limitada en banda y no se produzca el fenómeno de solapamiento de bandas de frecuencias (o "aliasing"), que conllevaría una pérdida de información de la señal, y por lo tanto podría llegar a inutilizar el sistema. En concreto, este filtro eliminará las componentes frecuenciales de la señal de entrada por encima de la frecuencia de Nyquist.

2.2.2 Muestreo

El muestreo es el procedimiento mediante el cual una señal analógica $x_a(t)$, continua en el tiempo, se transforma en una señal en tiempo discreto $x(n)$, formada por las "muestras" tomadas de la señal original en distintos instantes de tiempo discreto. El método de muestreo más utilizado es el muestreo uniforme (Figura 2.3), que toma las muestras en intervalos de tiempo equiespaciados (T_s):

$$x(n) = x_a(nT_s), \Leftarrow -\infty < n < +\infty \qquad (2.1)$$

Donde T_s es el periodo de muestreo y $F_s=1/T_s$ es la frecuencia de muestreo.

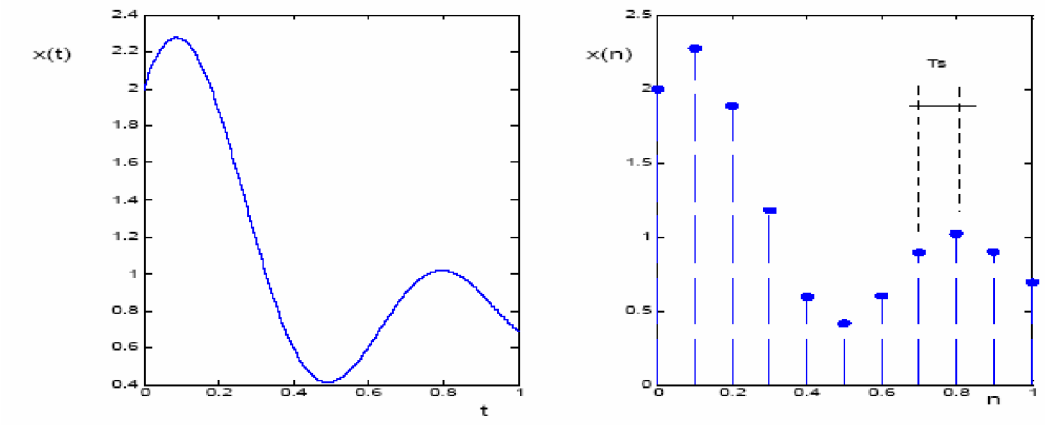


Figura 2.3.: Muestreo uniforme de una señal analógica

2.2.2.1 Teorema de muestreo

Teorema de muestreo:

"Si una señal continua, ***S(t)****, tiene una banda de frecuencia tal que* $\boldsymbol{f_m}$ *es la mayor frecuencia comprendida dentro de dicha banda, dicha señal podrá reconstruirse sin distorsión a partir de muestras de señal tomadas a una frecuencia* $\boldsymbol{f_s}$*, siendo* $\boldsymbol{2f_m < f_s}$*".*

La señal $x_a(t)$ se puede reconstruir totalmente a partir de sus muestras si la tasa de muestreo o frecuencia F_s cumple que $F_s > 2B$, siendo B la componente espectral máxima de la señal $x_a(t)$.

$F_N = 2B$ se denomina Tasa o Frecuencia de Nyquist.

Teorema de Muestreo $\Longrightarrow \quad F_s > F_N$

Si la frecuencia de muestreo F_s es inferior a la frecuencia de Nyquist F_N se produce un solapamiento del espectro de la señal (aliasing), lo que imposibilita una reconstrucción perfecta de la señal.

2.2.3 Cuantización

La cuantificación o cuantización convierte una señal en tiempo discreto, con la amplitud definida en un intervalo continuo, en una señal en tiempo discreto definida únicamente para un conjunto de valores discretos de amplitud. La señal discreta se limita a un número finito de amplitudes posibles, resultando una diferencia entre la señal cuantizada y la señal sin cuantizar denominada error de cuantización.

$$e_q(n) = x_q(n) - x(n)\ ,\ x_q(n) = Q[x(n)] \qquad (2.2)$$

Las amplitudes permitidas para la señal discreta se denominan niveles de cuantización. La resolución del cuantizador es la distancia entre dos niveles de cuantificación sucesivos (Δ), cuantos más niveles se usen (L) mayor precisión y menor error de cuantificación. En la figura 2.4. se representa la gráfica típica de un cuantizador de 8 niveles.

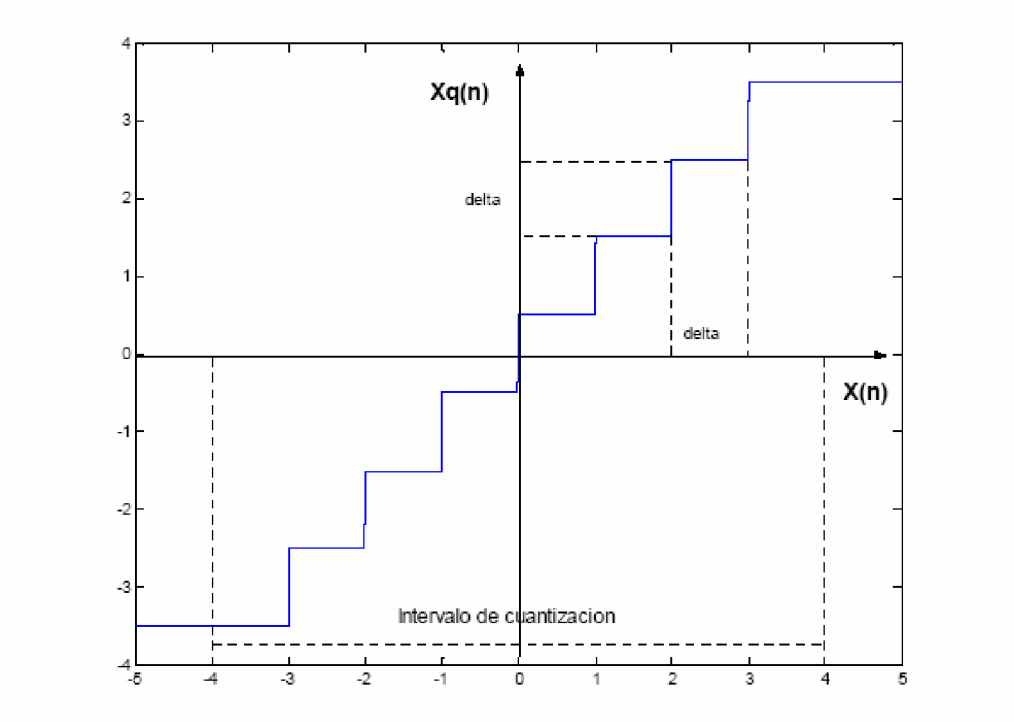


Figura 2.4.: Cuantizador de 8 niveles

2.2.4 Codificación

La codificación consiste en asignar a cada valor discreto de la señal cuantizada $x_q(n)$ una secuencia de *b* bits. El número de niveles de cuantización está fijado por los bits de precisión del codificador, y viceversa. Si disponemos de b bits en el codificador se pueden representar un total de 2^b números binarios, luego el número de niveles de cuantización máximo se define como $L = 2^b$.

Alternativamente, se puede decir que el número mínimo de bits del codificador debe cumplir la relación: $b \geq \log_2 L$

2.2.5 Conversión de la tasa de muestreo

El proceso de cambiar la frecuencia de muestreo de una señal digital a otra frecuencia dada se denomina conversión de la tasa de muestreo. Este proceso ofrece varias posibilidades para el factor de conversión:

- Diezmado o Submuestreo (1/D): Consiste en reducir la tasa de muestreo por un factor entero D. Produce un ensanchamiento del espectro de la señal digital de entrada respecto de la nueva frecuencia.
- Interpolación o Sobremuestreo (I): Incrementa la tasa de muestreo por un factor entero I. Produce una compresión del espectro de la señal digital de entrada respecto de la nueva frecuencia.
- Factor racional I / D: Es una combinación de los dos casos anteriores.

2.3 Convertidor A/D

En este capítulo hemos tratado aspectos fundamentales de la conversión analógica/digital sin considerar en ningún momento su arquitectura interna o diseño de circuito. Los convertidores han sido tratados como cajas negras tales que sólo se han discutido sus relaciones entre entrada y salida.

Definiremos a continuación ciertas figuras de mérito típicas de los CA/D: relación señal a ruido, rango dinámico, relación señal a ruido y distorsión, y número efectivo de bits.

2.3.1 Figuras de Mérito

En este sub-apartado veremos las figuras de mérito mencionadas anteriormente una a una:

2.3.1.1 Relación señal a ruido

Se define la relación señal-ruido (*SNR* del inglés "Signal to Noise Ratio") como el cociente entre la potencia de salida a la frecuencia de la entrada (supuesta ésta sinusoidal) y la potencia en banda del ruido total. Si tenemos en cuenta tan solo el ruido de cuantización y que la amplitud de la entrada sinusoidal al CA/D es A, la *SNR* es:

$$SNR_{dB} = 10\log\left(\frac{A^2/2}{P_e}\right) \tag{2.3}$$

donde *log* significa logaritmo en base diez. Vemos que a priori la *SNR* aumenta de forma monótona con el nivel de entrada. Pero esto sólo ocurre hasta un cierto valor, donde un exceso en el nivel de la señal de entrada del cuantizador provocará un aumento desmesurado en el ruido de cuantización (una vez la amplitud de la entrada sobrepase el fondo de escala del cuantizador), y por tanto una bajada abrupta de *SNR* a la salida del cuantizador.

2.3.1.2 Rango dinámico

Se define el rango dinámico (*DR* del inglés "Dinamic Range") como el cociente entre la potencia a la salida a la frecuencia de una sinusoide con una amplitud igual al rango completo del cuantizador y la potencia a la salida para una sinusoide que tenga la misma frecuencia y amplitud tal que no sea distinguible del fondo de ruido; esto es, con una *SNR*=0dB. Usualmente se expresa en dB.

Idealmente el rango dinámico de entrada viene limitado por la escala completa de entrada del cuantizador. Por otro lado la potencia de salida para una sinusoide de amplitud tal que su *SNR*=0dB vale P_e. Por tanto,

$$DR_{dB} = 10\log\left(\frac{(x_{FS}/2)^2}{2P_e}\right) \tag{2.4}$$

2.3.1.3 Relación señal a ruido y distorsión

Se define la relación señal a ruido y distorsión (*SNDR* del inglés "Signal to Noise and Distortion Ratio") como el cociente entre la potencia de salida a la frecuencia de la entrada (supuesta ésta sinusoidal) y la potencia en banda del ruido total y de la distorsión. Es decir,

$$SNDR_{dB} = 10\log\left(\frac{P_{signal}}{P_{noise} + P_{dist}}\right) \tag{2.5}$$

donde *log* significa logaritmo en base diez, P_{signal} representa la potencia de la señal, P_{noise} la potencia del ruido y P_{dist} la potencia de la distorsión.

2.3.1.4 Número efectivo de bits

Se define el número efectivo de bits de un CA/D (y se representa por ENOB) como:

$$ENOB(bits) = \frac{SNDR(dB) - 1.76}{6.02} \tag{2.6}$$

Capítulo 3
Convertidores A/D Pipeline

3.1 El convertidor pipeline

El convertidor A/D pipeline se compone de varias etapas, conteniendo cada una de ellas un convertidor analógico digital (CAD), un convertidor digital analógico (CDA), un restador y un amplificador de residuo. La última etapa necesitará sólo de un CAD. En ocasiones, se emplea un circuito de muestreo y retención (S&H) a la entrada para evitar errores por desviaciones debidas a retrasos en las dos señales de entrada al restador. Debido a su simplicidad y rapidez, el CAD suele ser un cuantizador Flash. Al conjunto del amplificador de residuo, el restador y el CDA se le llama también convertidor digital analógico multiplicador (MDAC) y desempeñará , además, la función de S&H. Un esquema general de un convertidor pipeline puede verse en la figura 3.1.

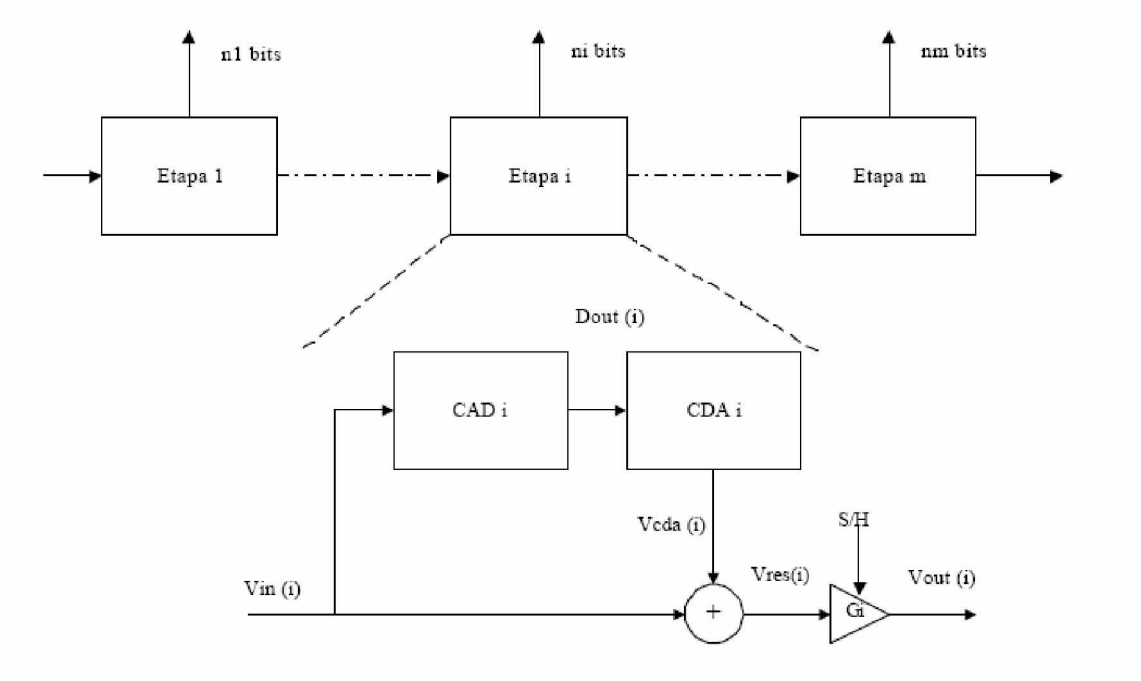


Figura 3.1.: El convertidor pipeline

Una etapa genérica i procesa la señal de entrada en dos fases y es responsable de la extracción de un conjunto j de bits del resultado final. En la primera fase, la tensión de entrada es muestreada y almacenada en el MDAC, mientras que el cuantizador Flash determina los j bits de la etapa. En la segunda fase, la tensión del residuo, obtenida mediante la resta de la reconstrucción llevada a cabo por el CDA de la cuantización gruesa y de la tensión mantenida de la entrada, es amplificada por una determinada ganancia. El resultado de esta amplificación actuará como entrada de la siguiente etapa, donde se llevará a cabo un proceso idéntico al explicado.

Normalmente, el número de bits N_i es potencia de dos, es decir,

$$N_i = 2^{ni} \tag{3.1}$$

donde n_i es el número de bits de la etapa i.

Los convertidores pipeline se han convertido en los más utilizados en el diseño de arquitecturas de resolución moderada (8 a 12 bits) y frecuencias de muestreo del orden de decenas de megahercios. Para frecuencias de muestreo menores se suelen utilizar convertidores de aproximaciones sucesivas (bajas resoluciones) o arquitecturas Sigma-Delta (altas resoluciones), mientras que para frecuencias superiores (cientos de megahercios) continúan dominando los convertidores flash y sus variantes.

Como ilustración del método de diseño de las etapas de un CA/D pipeline, veremos un ejemplo con 10 bits. Como regla general, suele aplicarse que, para resoluciones altas, del orden de 15 bits, se escogen resoluciones de 4 o 5 bits para las dos primeras etapas y 2 bits para las restantes, y para convertidores de hasta 10 bits de resolución se utilizan resoluciones bajas (2 bits) para todas sus etapas. Por tanto, en nuestro caso, dividiremos el convertidor en nueve etapas de dos bits de resolución cada una.

En la figura 3.2. se puede observar el diagrama de bloques general del convertidor.

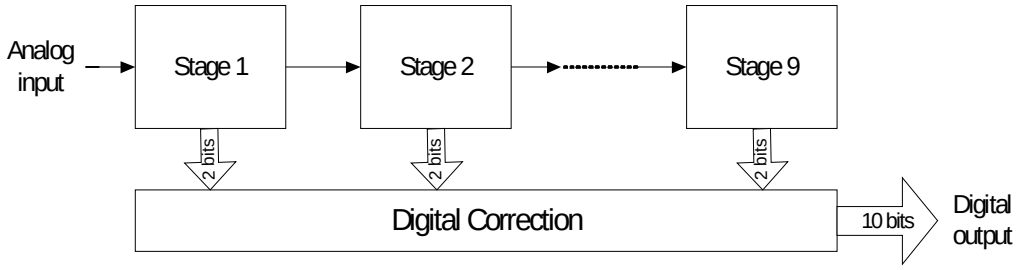


Figura 3.2.: Diagrama de bloques del convertidor

Cada una de las etapas realiza una conversión con dos bits de resolución y transmite el residuo a la siguiente etapa. Los 18 bits resultantes de las 9 conversiones se combinan mediante la corrección digital para obtener los 10 bits buscados. La redundancia existente permite la utilización de convertidores de 1.5 bits de resolución y simplifica enormemente el diseño de los comparadores. Este tipo de convertidores compuestos de etapas de 1.5 bits de resolución se suelen llamar convertidores RSD (Redundant Signed Digit).

Como puede verse en la figura 3.3, cada una de las etapas que componen el convertidor (a excepción de la última) está compuesta por un ADC, un DAC, un SHA (Sampling-and-Hold Amplifier) y un circuito de resta para el cálculo de residuo.

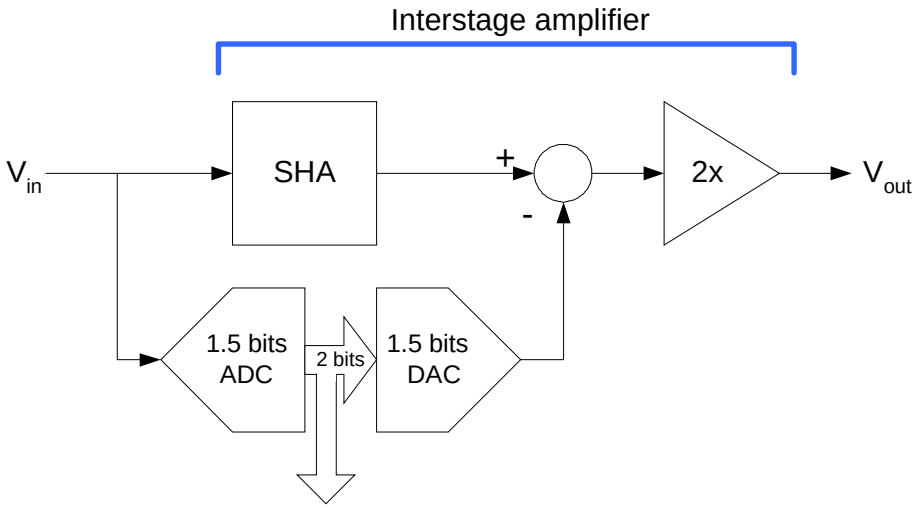


Figura 3.3. Arquitectura de una etapa pipelined.

3.2 Principales no idealidades en un CA/D Pipeline

Las no idealidades más importantes en un CA/D pipeline, que analizaremos una a una en los siguientes sub-apartados, son:

- Ganancia DC Finita del amplificador operacional.
- Tamaño de las capacidades.
- Slew Rate y Ancho de Banda del amplificador operacional.
- Offset en los DACs.
- Jitter.

3.2.1. Ganancia DC Finita del amplificador operacional

En el presente sub-apartado se estudia la influencia de la ganancia finita del amplificador operacional sobre la resolución del convertidor. Para desarrollar el modelo se ha utilizado el circuito de la figura 3.4., en la fase de cálculo del residuo.

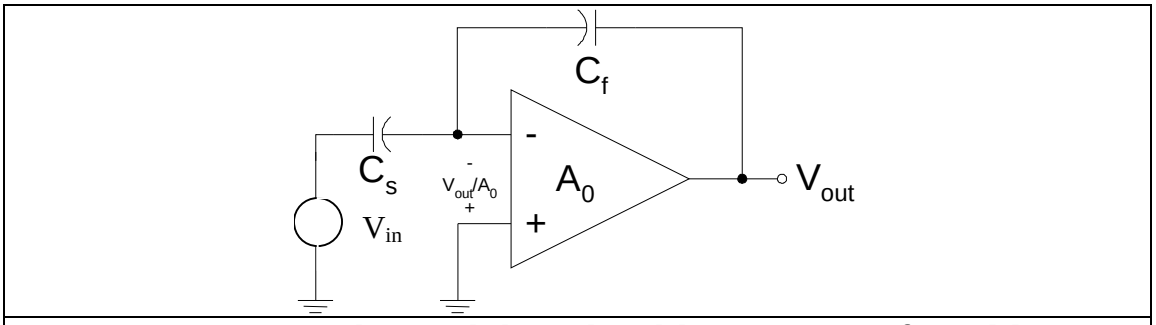


Figura 3.4. Circuito usado para deducir el modelo con ganancia finita del op-amp

Debido a la ganancia finita del amplificador (A_0), la ganancia del amplificador de residuo se ve modificada según la siguiente expresión:

$$V_{out} = \frac{2}{1+\frac{2}{A_0}} V_{in} \qquad (3.2)$$

Esta desviación de la ganancia del amplificador de residuo no es corregida por la corrección digital.

3.2.2. Tamaño de las capacidades

El tamaño de las capacidades utilizadas en el cálculo del residuo determina el consumo del amplificador operacional, y por tanto, del convertidor completo. Por esa razón, las capacidades deben ser lo menores posible. Sin embargo, existen dos factores que limitan el tamaño de estas capacidades:

3.2.2.1 Ruido térmico

En un circuito de capacidades conmutadas el ruido introducido durante el proceso de muestreo es inversamente proporcional al tamaño de la capacidad. Basándonos en este principio, y teniendo en cuenta que el ruido máximo admitido viene dado por la resolución del convertidor, es posible calcular la capacidad de muestreo mínima.

La potencia total de ruido debida al proceso de muestreo viene dada por la siguiente ecuación:

$$\overline{v_{out}^2} = \frac{kT}{C_s} \tag{3.3}$$

Por otra parte, la potencia correspondiente al ruido de cuantización para una entrada senoidal, bajo supuesto de ruido blanco, puede expresarse como:

$$\varepsilon_q^2 = \frac{\Delta^2}{12} \tag{3.4}$$

donde $\Delta = V_{REF} / 2^n$ se corresponde con el tamaño del escalón de cuantización de un convertidor de n bits y V_{REF} representa la tensión de fondo de escala.

La relación señal a ruido a la salida para una entrada analógica senoidal de amplitud $\frac{V_{REF}}{2}$, expresada en decibelios, viene dada por la expresión:

$$\text{SNR}=6.02\ n+1.76 \tag{3.5}$$

donde n representa, como se ha mencionado, el número de bits de resolución. En nuestro caso, SNR=61.96 dB. Si asumimos una desviación de 1 dB respecto a la relación señal a ruido ideal, la capacidad de carga permitida a la salida del amplificador será de $C_L \geq 49.91$ fF, valor suficientemente bajo para concluir que el ruido térmico no es un factor determinante a la hora de fijar el tamaño mínimo de las capacidades empleadas en el cálculo del residuo.

3.2.2.2 Desapareamiento entre capacidades

Cualquier variación en el cociente C_f/C_s producirá una desviación en la ganancia del residuo, que no se compensaría en la corrección digital. Por otro lado, mientras más pequeña sean las capacidades peor es el apareamiento entre ellas. En nuestro caso, el mínimo valor de las capacidades vendrá limitado por este efecto.

Para desarrollar el modelo utilizado se ha tenido en cuenta que $V_{ref} = 2V_{DAC}$, siendo V_{ref} la tensión de referencia y V_{DAC} la salida del DAC en el modelo ideal.

El desapareamiento entre capacidades se medirá mediante el parámetro ε, que viene dado por la siguiente ecuación:

$$\frac{C_s}{C_f} = 1 + \varepsilon \tag{3.6}$$

La tensión de salida considerando el desapareamiento entre capacidades queda:

$$V_{out} = 2(V_{in} - V_{DAC}) + \varepsilon (V_{in} - 2V_{DAC}) \tag{3.7}$$

3.2.3. Slew Rate y Ancho de Banda del amplificador operacional.

En este sub-apartado estudiaremos la influencia de la velocidad del amplificador operacional en la resolución del convertidor, definiendo las especificaciones dinámicas del opamp, es decir, su slew rate y ancho de banda.

Veremos el efecto del slew rate y el ancho de banda mediante el circuito de la figura 3.5., en la fase de amplificación del residuo, en la cual la carga almacenada en C_s pasa a C_f. En la práctica, este traspaso de carga nunca se realiza completamente, no alcanzando la salida del MDAC el valor deseado, debido a las limitaciones dinámicas del amplificador.

El proceso de carga estará limitado por dos efectos. Por un lado, si el cambio de tensión a la salida es muy abrupto el amplificador saldrá de su zona de funcionamiento lineal, estando la pendiente limitada por el slew rate. Por otro lado, cuando el amplificador está funcionando linealmente (asumimos un comportamiento de primer orden) la carga de la capacidad se producirá siguiendo una característica exponencial.

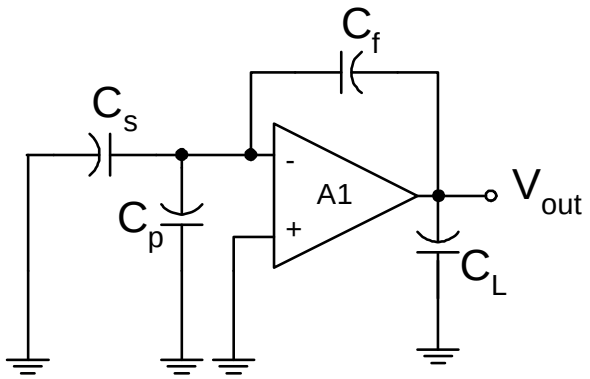


Figura 3.5. Modelo para calcular la influencia del SR y BW del amplificador operacional en cálculo del residuo.

Asumimos un amplificador con respuesta en frecuencia de primer orden dada por:

$$A(s) = \frac{A_0}{1 - \frac{s}{p_1}} \tag{3.8}$$

Donde, A_0 es la ganancia DC del amplificador y p_1 la frecuencia del polo, definidos como:

$$A_0 = g_m r_{out} \tag{3.9}$$

$$p_1 = -\frac{1}{r_{out} C_{Leff}} \tag{3.10}$$

La capacidad C_{Leff} es la capacidad efectiva que ve el amplificador en bucle abierto a la salida. Es decir:

$$C_{Leff} = C_L + \frac{C_f (C_p + C_s)}{C_f + C_p + C_s} \tag{3.11}$$

El factor de realimentación viene dado por el divisor capacitivo compuesto por C_f y C_s:

$$\beta = \frac{C_f}{C_s + C_f + C_p} \approx \frac{1}{2} \tag{3.12}$$

Para conocer la respuesta transitoria del sistema calculamos la función de transferencia en bucle cerrado:

$$A_{cl}(s) = \frac{A}{1 - A\beta} = \frac{r_{out} g_m}{1 + r_{out} C_{Leff} s - \frac{C_f}{C_f + C_p + C_s} g_m r_{out}} \approx \frac{g_m}{C_{Leff} s - \frac{C_f}{C_f + C_p + C_s} g_m} \tag{3.13}$$

Al ser un sistema de primer orden, la constante de tiempo se corresponde con la inversa de la frecuencia del polo:

$$\tau = 2\frac{C_{Leff}}{g_m} \tag{3.14}$$

Nótese que se corresponde con la constante de tiempo de un sistema con ancho de banda la mitad del producto ganancia ancho de banda del amplificador con una capacidad de carga C_{Leff}.

Es posible hacer un estudio más detallado del proceso de carga del condensador teniendo en cuenta el slew rate del amplificador. La ecuación que rige el comportamiento del MDAC es la siguiente:

$$V_{out} = V_{in} + g\left(V_{in} - 2V_{DAC}\right) \tag{3.15}$$

Siendo *g(x)* la función que modela el efecto del *slew-rate* y el ancho de banda del amplificador sobre el proceso de carga, y tiene la siguiente forma:

$$g(x) = \begin{cases} x\left(1 - e^{-\frac{T_S}{2\tau}}\right) & |x| \le \tau\zeta \\ x - \operatorname{sgn}(x)\tau\zeta\, e^{\left(\frac{|x|}{\tau\zeta} - \frac{T_s}{2\tau} - 1\right)} & \tau\zeta < |x| \le (\tau + T_s)\zeta \\ \operatorname{sgn}(x)\zeta\,\frac{T_s}{2} & (\tau + T_s)\zeta < |x| \end{cases} \tag{3.16}$$

Donde τ representa el tiempo de establecimiento, ζ el *slew-rate* del amplificador y T_s el tiempo de muestreo.

3.2.4. Offset en los DACs

El efecto del offset que está presente en los DACs es un cambio en los niveles de decisión, que ya no estarán equidistribuidos a lo largo del rango que va de –FS/2 a FS/2 (suponiendo un convertidor simétrico).

3.2.5. Jitter

El ruido jitter está asociado al proceso de muestreo de señales analógicas para trabajar con señales en tiempo discreto. El mecanismo del jitter no está demasiado claro (se sabe que da como resultado un tiempo de muestreo no uniforme) y, por tanto, no hay muchas técnicas o diseños que traten el problema. Aumentar la resolución y la velocidad de los convertidores depende en gran medida de reducir el jitter.

El ruido jitter produce un error que incrementa la potencia total del error a la salida del cuantizador. La magnitud de este error es función de las propiedades estadísticas del jitter. El error introducido cuando una señal sinusoidal x(t) de amplitud A y frecuencia f_{in} es muestreada en un instante (dado con un error de δ) viene dado por:

$$x(t+\delta) - x(t) \approx 2\pi f_{in}\delta A\cos(2\pi f_{in}t) = \delta\frac{d}{dt}x(t) \tag{3.17}$$

Capítulo 4
Convertidores A/D Flash

4.1 El convertidor flash

El convertidor A/D flash tiene la topología más simple de todos los convertidores, y además es el más rápido. La técnica flash es también conocida como "la aproximación en paralelo".

El convertidor flash alcanza grandes velocidades de conversión debido a que usa una matriz de comparadores en paralelo que muestrean la señal analógica simultáneamente. Como requerimos un comparador por cada nivel de cuantización que tengamos, el número de comparadores se dobla por cada bit adicional de resolución que queramos tener. Así pues, el problema que surge con esta técnica es el significativo incremento de disipación de potencia en comparación con otras técnicas de conversión A/D.

Un convertidor flash de N bits requiere una matriz de $2^N - 1$ comparadores. La entrada analógica se conecta a una entrada de la matriz de comparadores, mientras que la otra entrada de cada comparador se conecta a tensiones de referencia fijas. Estas referencias representan $2^N - 1$ niveles de tensión equidistantes correspondientes a los $2^N - 1$ puntos de conmutación entre los extremos del rango del convertidor.

La desventaja de todo esto, evidentemente, es que se necesitan muchos comparadores, resistencias para crear las referencias de tensión, e interconexiones, y además que se incrementarán exponencialmente con el incremento de resolución del convertidor.

En la figura 4.1. puede verse la arquitectura típica de un convertidor A/D flash:

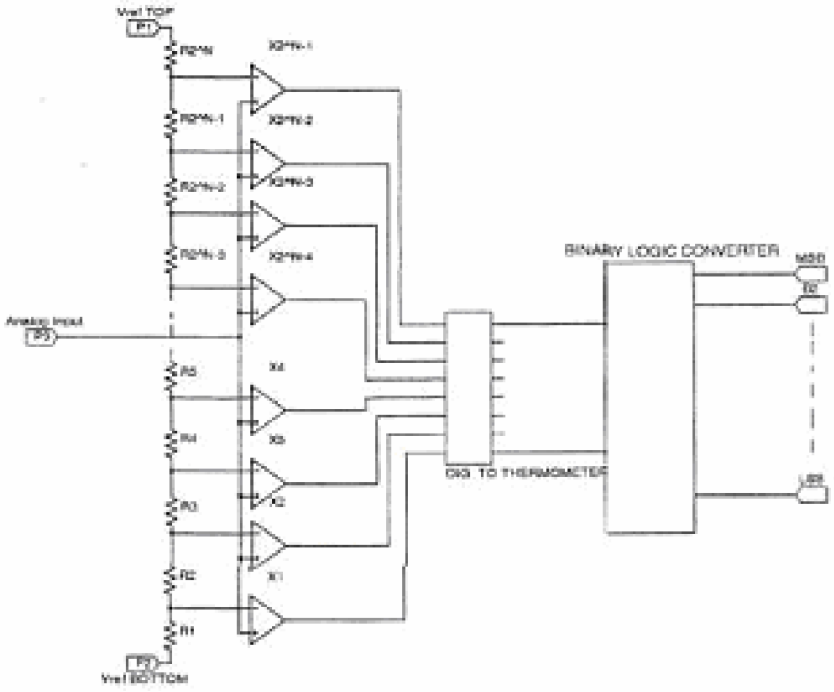


Figura 4.1.: Convertidor A/D flash típico

La arquitectura anterior presenta 2^N resistencias en la escalera de referencia, proporcionando (en el caso ideal, es decir, si las resistencias fuesen todas idénticas) $2^N - 1$ niveles de tensión equidistantes a los comparadores. La diferencia de tensiones entre 2 resistencias adyacentes es igual a:

$$V_{LSB} = \frac{V_{REF_TOP} - V_{REF_BOT}}{2^N} \tag{4.1}$$

donde V_{REF_TOP} y V_{REF_BOT} representan las 2 referencias de tensión proporcionadas al convertidor.

El modo de operación del convertidor es el siguiente: cada comparador cuya entrada analógica esté por debajo de su referencia pone un "0" lógico a su salida, y los comparadores cuya entrada supere la referencia pone un "1". Estas salidas sirven como entrada a un decodificador termométrico, que detecta la transición de ceros lógicos a unos en la matriz de comparadores. Esta transición, en teoría, debe ser única. La siguiente operación ya sería la decodificación de termométrico a binario, obteniéndose la palabra binaria de salida.

4.2 Principales no idealidades en un CA/D Flash

Las no idealidades más importantes en un CA/D flash son las siguientes:

- Offset en las tensiones de referencia, debidas a desapareamientos entre las resistencias que proporcionan a la matriz de comparadores dichas tensiones de referencia.

- Offset en los comparadores, que provocarán que tenga que haber una diferencia (positiva o negativa) entre la entrada analógica y la tensión de referencia de cada comparador en la comparación (en el caso ideal, se comparaba si era mayor o menor, ahora habrá que tener en cuenta el offset). Este offset, si es suficientemente grande en algún comparador, podría provocar que la transición de ceros a unos en la escalera de comparadores no fuese única, con lo cual el código termométrico y su posterior decodificación podrían fallar.

- Histéresis en los comparadores, que puede hacer que la salida de un comparador en concreto no pase de "0" a "1" y viceversa cuando se espera, cuanto más grande sea esta histéresis con más probabilidad podrá ocurrir este fenómeno.

Capítulo 5
Interfaces gráficas de usuario en Matlab

5.1 Introducción

En el presente apartado se describirá brevemente el proceso a seguir para la creación de una interfaz gráfica de usuario (GUI, del inglés "Graphical User Interface") en Matlab.

A grandes rasgos, la creación de una GUI sigue 2 pasos básicos, en los que entraremos en profundidad en los siguientes apartados:

- Diseño (lay out) de la GUI: Se realizará en un editor de layout, haciendo clic y arrastrando al área de layout componentes de la GUI (como paneles, botones, campos de texto, barras deslizantes, menús, etc.).

- Programación de la GUI: Al terminar el diseño de la GUI, se genera automáticamente un archivo .m que controla el funcionamiento de la GUI. Dicho archivo inicializa la GUI y proporciona un entorno para todas las callbacks (comandos que se ejecutan cuando un usuario hace clic sobre un componente de la GUI) de la GUI. Usando el editor de archivos .m, podremos añadir código a las callbacks para que realicen las funciones deseadas.

5.2 Diseño de la GUI

En el presente apartado vamos a explicar cómo se diseña una GUI en Matlab. En general, para diseñar una GUI tendremos que dar 2 pasos, que detallaremos en los siguientes sub-apartados:

- Creación o apertura de la interfaz de usuario: Necesitamos crear una GUI nueva o cargar una ya existente.

- Edición de la interfaz de usuario: Una vez realizado el paso anterior, podemos editar el aspecto de la GUI.

5.2.1. Creación o apertura de la interfaz de usuario

Una vez abierto Matlab y situados en la ventana de comandos, tenemos que ejecutar el asistente para creación o apertura de interfaces de usuario, denominado GUIDE. Este asistente se arrancará con la orden *guide*, cuyo resultado será la aparición de la ventana que se muestra en la figura 5.1.

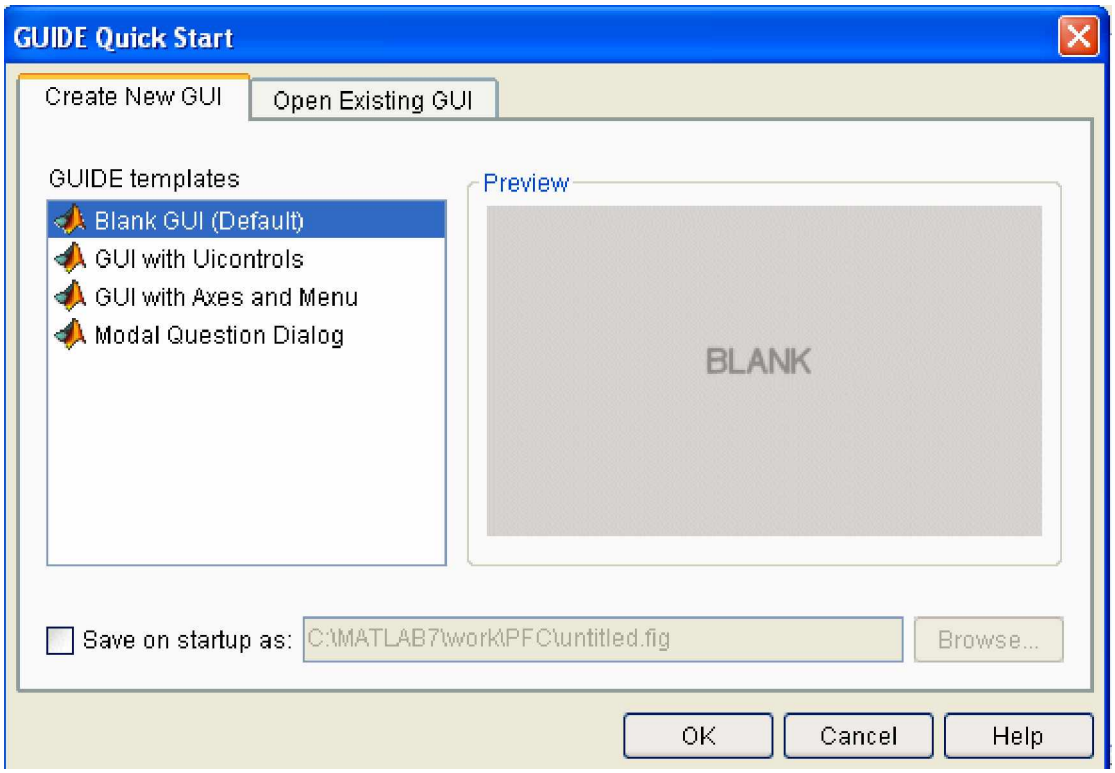


Figura 5.1.: Ventana de inicio de creación de GUI

En la figura 5.1. puede apreciarse que, en la parte superior, aparecen 2 pestañas, que serán las 2 funciones distintas que puede realizar inicialmente el asistente. Dichas funciones, que veremos con más detalle en los siguientes sub-apartados, son:

- Crear una nueva GUI.
- Abrir una GUI ya existente.

GUIDE abrirá la GUI en el editor de layout al finalizar la creación o apertura de la misma. En la parte inferior de la figura 5.1. puede verse que existe la posibilidad de seleccionar "Save on startup as" y escribir en el campo que aparece a la derecha el nombre del archivo en el que GUIDE guardará la GUI antes de abrirla en el editor de layout. Si se elige no salvar la GUI en este punto, GUIDE pedirá que se guarde la primera vez que se ejecute la GUI.

En cuanto a los archivos que se generan, señalar que GUIDE almacena cada GUI en 2 archivos, que se generarán, como se ha mencionado anteriormente, la primera vez que se guarde o se ejecute la GUI:

- Un archivo .fig con la descripción completa del diseño de la GUI y de sus componentes: push buttons, menus, ejes, etc.
- Un archivo .m que contendrá el código que controla la GUI, incluyendo todas las callbacks para cada uno de sus componentes.

5.2.1.1 Creación de una nueva GUI

Eligiendo esta opción, ahora tenemos que decidir el tipo de GUI, existiendo 4 posibilidades predeterminadas entre las que podremos elegir:

- GUI en blanco.
- GUI con Uicontrols.
- GUI con Ejes y Menú.
- Modal Question.

5.2.1.1.1 GUI en blanco:

La GUI en blanco predeterminada que se mostrará en el editor de layout puede verse en la figura 5.2. Se recomienda elegir una GUI en blanco sólo si el resto de GUIs predeterminadas no son puntos de partida adecuados para el diseño deseado.

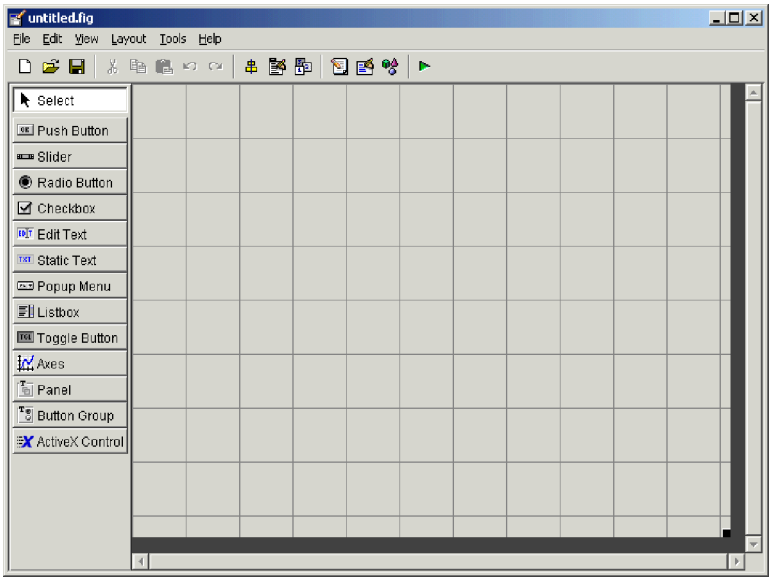


Figura 5.2.: GUI en blanco predeterminada

5.2.1.1.2 GUI con Uicontrols:

La GUI con Uicontrols predeterminada que se mostrará en el editor de layout puede verse en la figura 5.3.

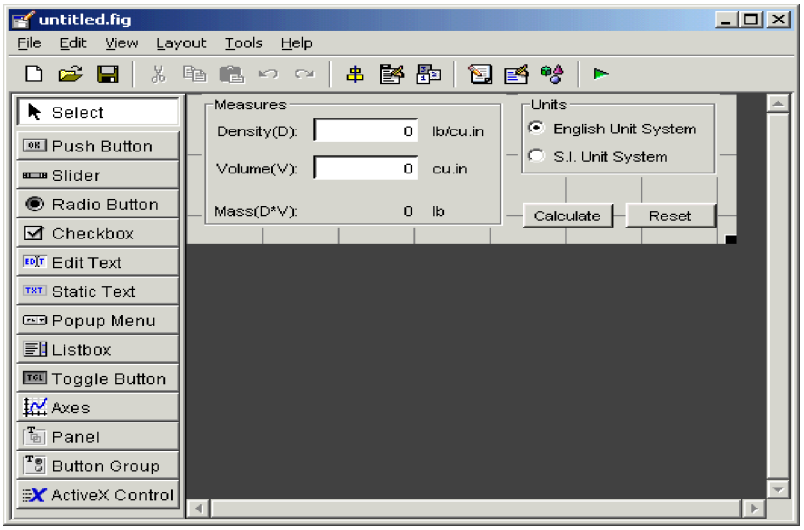


Figura 5.3.: GUI con Uicontrols predeterminada

Cuando se ejecuta la GUI haciendo clic en el icono de ejecución , la GUI aparece tal como puede verse en la figura 5.4.

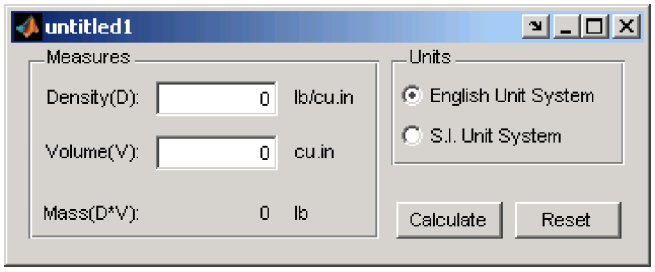


Figura 5.4.: Ejecución GUI con Uicontrols predeterminada

5.2.1.1.3 GUI con Ejes y Menú:

La GUI con Ejes y Menú predeterminada que se mostrará en el editor de layout puede verse en la figura 5.5.

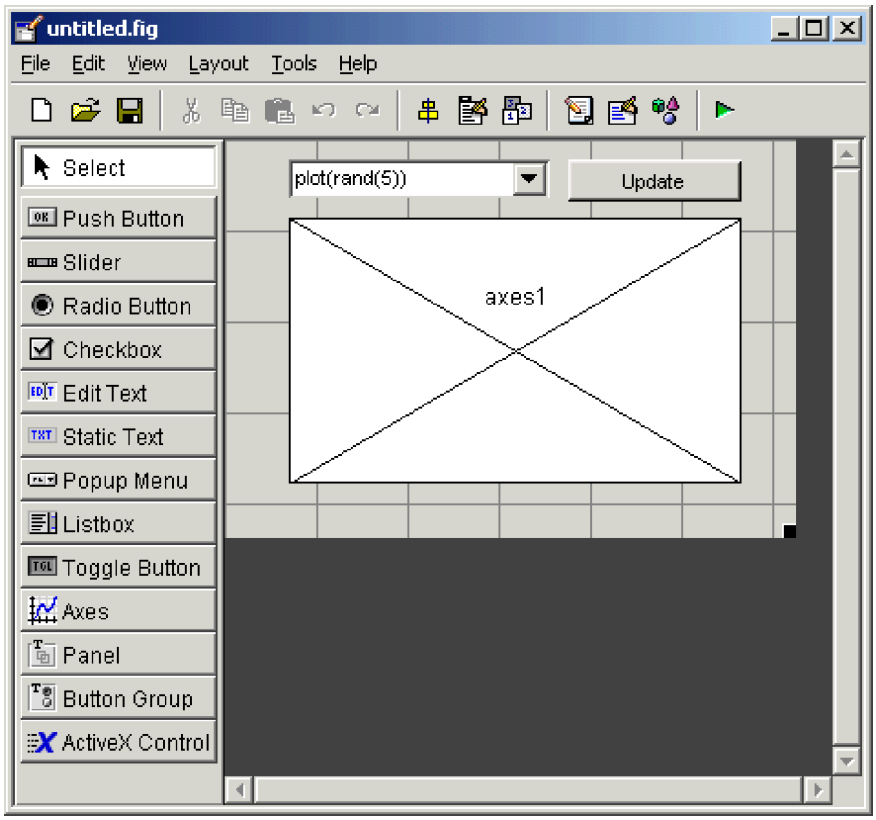


Figura 5.5.: GUI con Ejes y Menú predeterminada

Cuando se ejecuta la GUI haciendo clic en el icono de ejecución , la GUI aparece tal como puede verse en la figura 5.6.

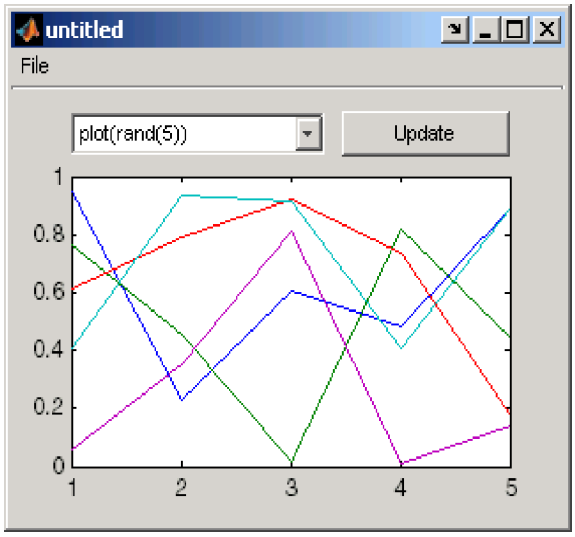


Figura 5.6.: Ejecución GUI con Ejes y Menú predeterminada

5.2.1.1.4 Modal Question:

La GUI de tipo Modal Question predeterminada que se mostrará en el editor de layout puede verse en la figura 5.7.

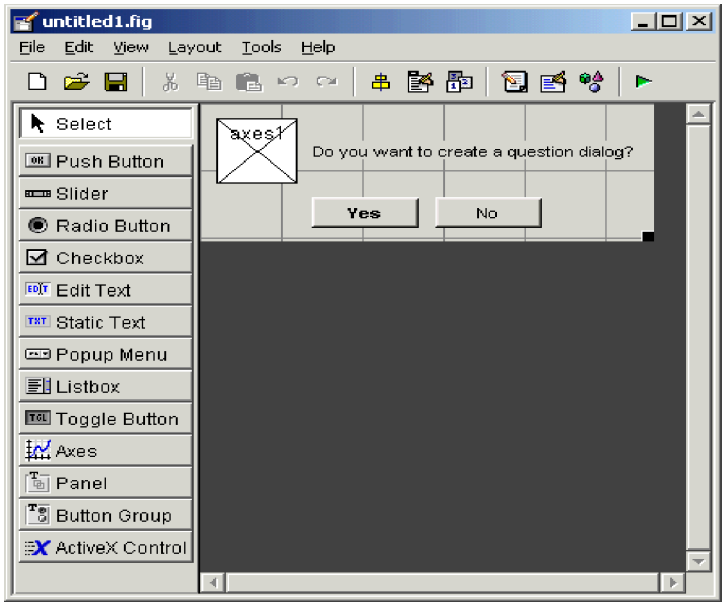


Figura 5.7.: GUI tipo Modal Question predeterminada

La ejecución de esta GUI produce como resultado el cuadro de diálogo que aparece en la figura 5.8.

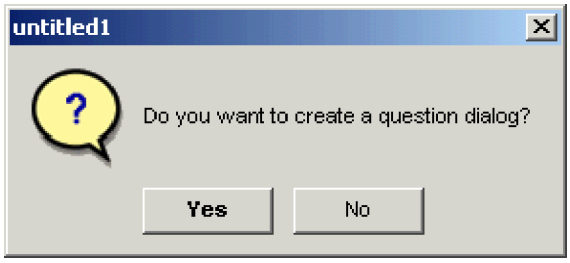


Figura 5.8.: Ejecución GUI tipo Modal Question predeterminada

Cabe destacar que este tipo de GUI es modal, es decir, el usuario no puede interactuar con otras ventanas de Matlab hasta que haga clic sobre uno de los botones del cuadro de diálogo.

5.2.1.2 Apertura de una GUI ya existente

Eligiendo esta opción, nos aparece un listado de las GUIs recientemente abiertas, como puede verse en la figura 5.9. (en este caso, no aparece ninguna porque partimos de que no hemos abierto ninguna GUI antes, en caso contrario aparecerían dichas GUIs). Puede elegirse una de estas GUIs, o bien hacer clic en el botón "Browse…" y seleccionar otra GUI para añadirla al listado mencionado.

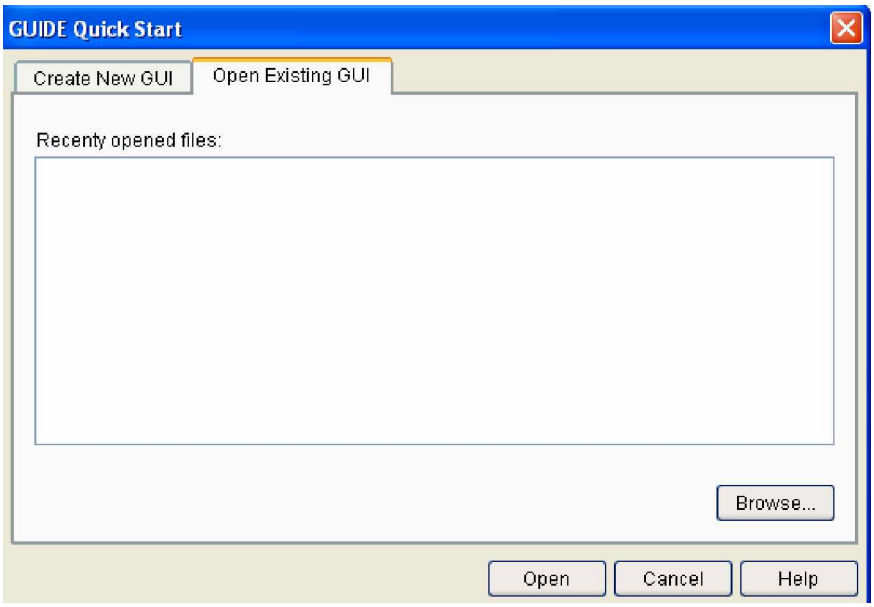


Figura 5.9.: Apertura de una GUI ya existente

5.2.2. Edición de la interfaz de usuario

Una vez creada una nueva GUI o abierta una ya existente, el siguiente paso a realizar será editar la interfaz de usuario. Esto se hará colocando componentes, como push buttons, pop-up menús, ejes, etc., desde la paleta de componentes, en el lado izquierdo del editor de layout, hasta el área de layout. Esta colocación de componentes puede efectuarse de 2 formas diferentes:

- Arrastrando el componente al área de layout y soltándolo allí.

- Seleccionando el componente en la paleta de componentes. De esta forma, el cursor pasa a ser una cruz. Ahora tenemos 2 posibilidades: colocar el cursor en el área de layout en el lugar en el que se desea que esté la esquina superior izquierda del componente y hacer clic (pone un tamaño predeterminado para el componente); o bien, colocar el cursor en el área de layout en el lugar en el que se desea que esté la esquina superior izquierda del componente, posteriormente establecer el tamaño del componente haciendo clic y arrastrando el cursor hasta la esquina inferior izquierda del componente antes de soltar el botón del ratón.

A continuación, vamos a describir brevemente los componentes que podremos encontrar en la paleta de componentes:

- Push Button: Genera una acción determinada cuando se hace clic sobre él (por ejemplo, un botón OK debe cerrar un cuadro de diálogo y aplicar una configuración determinada).

- Toggle Button: Genera una acción e indica si está activado (con un recuadro alrededor del nombre) o desactivado (nombre no recuadrado). Varios toggle buttons pueden ser excluyentes entre sí (a esta propiedad se le llama exclusión mutua).

- Radio button: Son botones que, relacionados en grupos, se excluyen mutuamente, es decir, únicamente puede seleccionarse un radio button de cada grupo de ellos. El estado activo del radio button se indica con un círculo relleno, el estado inactivo con un círculo vacío.

- Check box: Genera una acción cuando está activo (tic dentro del cuadro), y se indica como desactivado si no hay tic dentro del cuadro. Un check box es útil para dar al usuario varias opciones independientes entre sí, que se elegirá si hacerlas (activar el check box) o no hacerlas (dejarlo inactivo).

- Edit text: Es un campo que permite al usuario introducir o modificar cadenas de texto.
- Static text: Muestra líneas de texto. Este componente se usa típicamente para etiquetar otros componentes. El usuario no puede cambiar un static text interactivamente, y tampoco hay ninguna manera de invocar la callback asociada a este componente.

- Slider: Acepta una entrada numérica con un rango específico, permitiendo al usuario mover una barra deslizante, llamada slider o thumb. El usuario puede mover la barra de 3 formas distintas: presionando el botón del ratón y arrastrando la barra; o bien haciendo clic en el camino que sigue la barra; o bien haciendo clic en unas flecha arriba o abajo. La localización de la barra indica un porcentaje del rango especificado.

- List Box: Muestran una lista de elementos y permite a los usuarios elegir uno o más elementos de dicha lista.

- Pop-Up Menu: Es un menú que se despliega cuando el usuario hace clic en una flecha, mostrando una lista de opciones.

- Axes: Permite a la GUI mostrar gráficas e imágenes.

- Panel: Agrupa componentes de la GUI relacionados entre sí, permitiendo que la interfaz de usuario sea más sencilla de entender.

- Button Group: Al igual que un componente tipo panel, agrupa otros componentes, pero también pueden ser usados para gestionar la exclusión mutua de radio buttons y toggle buttons.

- ActiveX Component: Permiten mostrar controles de tipo ActiveX en la GUI.

5.3 Programación de la GUI

En el presente apartado se describirán los pasos básicos para comenzar la programación de una GUI en Matlab.

Una vez concluido el diseño de la GUI, para programar su funcionamiento hemos de usar el editor de archivos .m, al que se accede haciendo clic en el icono que aparece en la barra de herramientas del editor de layout.

Recordemos que GUIDE genera automáticamente un archivo .m a partir del diseño de la GUI que hemos realizado, y que la generación de este archivo se produce la primera vez que se guarda o se ejecuta la GUI.

El contenido de un archivo .m puede resumirse en 3 puntos básicos:

- Código de inicialización de la GUI.
- Código para implementar tareas previas a que la GUI aparezca en pantalla, tales como creación de datos, gráficas o imágenes.
- Código de las callbacks que se ejecutarán cada vez que el usuario haga clic en un componente de la GUI.

Inicialmente, en el archivo .m que GUIDE genera automáticamente, cada callback contiene sólo su línea de definición de función. Usando el editor de archivos .m, puede escribirse código en dichas funciones que haga que al hacer clic en cada componente, se obtenga el resultado deseado.

Puede verse la callback para cualquiera de los componentes de la GUI haciendo clic en el icono de función que se encuentra en la barra de herramientas del editor de layout. Haciendo clic en una callback de la lista que aparece, se muestra la sección del archivo .m que contiene la callback seleccionada, pudiendo editarse.

Capítulo 6
Operaciones en Matlab

6.1 Introducción

En el presente apartado se describirá brevemente la estructura del programa implementado (funciones que se llaman, etc.). En el apéndice 2 se halla reflejado el listado del código comentado completo del programa.

El nombre que se le ha dado al simulador de convertidores es *simconverter* (ésta será la orden que tendremos que teclear desde la ventana de comandos de Matlab para arrancarlo, ya que el script principal es el archivo simconverter.m). Este script contiene las llamadas a los dos scripts principales que ya contendrán el código del programa.

A grandes rasgos, estos scripts son:

- Initial: Se encarga de la declaración de las variables globales del programa. Dichas variables globales se han usado para poder disponer de los valores de los parámetros de interés del programa en el WorkSpace externo, de forma que desde el exterior del programa el usuario pueda ver el valor de estas variables, y editarlas desde la ventana de comandos. Este script se encuentra en el archivo initial.m.

- Guiproyecto: Este script contiene todo el código que controla el funcionamiento de la GUI. Dicho archivo, denominado guiproyecto.m, inicializa la GUI, contiene todas las callbacks (comandos que se ejecutan cuando un usuario hace clic sobre un componente de la GUI) de la GUI, y realiza todas las llamadas a funciones y scripts auxiliares necesarias para la implementación deseada en cada caso (entraremos en detalle en las funciones auxiliares a las que llama este script en el siguiente apartado).

6.2 Script de la GUI: Funciones y scripts auxiliares

Las funciones y scripts auxiliares a las que llamará el script de la GUI son las siguientes:

- INLDNL_matlab: Script que calcula INL, DNL y los códigos perdidos.

- CalcINLDNL: Función que recibe como parámetros el vector de salida del convertidor y el manejador de la GUI, y ejecuta el script INLDNL_matlab para calcular INL, DNL y los códigos perdidos.

- CalcSNDR: Función que recibe como parámetros el vector de salida del convertidor, el ancho de banda, la banda mínima de señal, el número de muestras que consideramos como banda de señal (del centro a un extremo), el manejador de la GUI y el manejador del objeto figura. Calculará SNDR, SNR y ENOB.

- SimulacionNoIdeal: Función que gestiona los análisis simples, es decir, realiza una única simulación para unos valores concretos de los parámetros. Realiza sus cálculos usando las funciones CalcINLDNL y CalcSNDR.

- Funciones de barrido: Funciones que gestionan los barridos simples y dobles, tenemos una función por cada barrido simple o doble que se haya implementado. La característica principal de las funciones de barrido es que su nombre siempre comienza por la palabra “Estudio”. Usará para realizar sus cálculos las funciones CalcINLDNL y CalcSNDR.

Capítulo 7
Librerías y modelos

7.1 Introducción

En el presente capítulo se describirán brevemente las librerías que se han implementado para que el usuario pueda hacer uso de ellas, y los modelos que también se le facilitan para el estudio de diversos convertidores. Dedicaremos a ello los dos próximos apartados.

7.2 Librerías

Las librerías que se han creado para el simulador se encuentran en la carpeta Library, agrupadas en 4 archivos (de los cuales los 3 primeros contienen librerías para convertidores pipeline, y el último para flash):

- Mylibrary.mdl: Contiene el generador de señal, y las etapas correspondientes tanto ideales como no ideales. También contiene un generador de ruido jitter.
- Substages.mdl: Contiene las subetapas: ADC, DAC y función para generar la no idealidad de Slew Rate y Ancho de Banda.
- Digcorrect.mdl: Contiene las posibles correcciones digitales para convertidores pipeline en el rango de 3 a 15 bits.
- Parts6bits.mdl: Contiene todos los bloques necesarios para el estudio de un convertidor flash de 6 bits.

Vemos estas librerías con más detalle en los siguientes sub-apartados.

7.2.1. Mylibrary.mdl

El aspecto de esta librería puede verse en la figura 7.1. Se aprecia que a las distintas etapas posibles se les ha dotado de diferentes colores (mostaza para etapa ideal, azul para etapa con ganancia finita, roja para etapa con desapareamiento entre las capacidades, etc.) para hacer más intuitiva la realización de modelos.

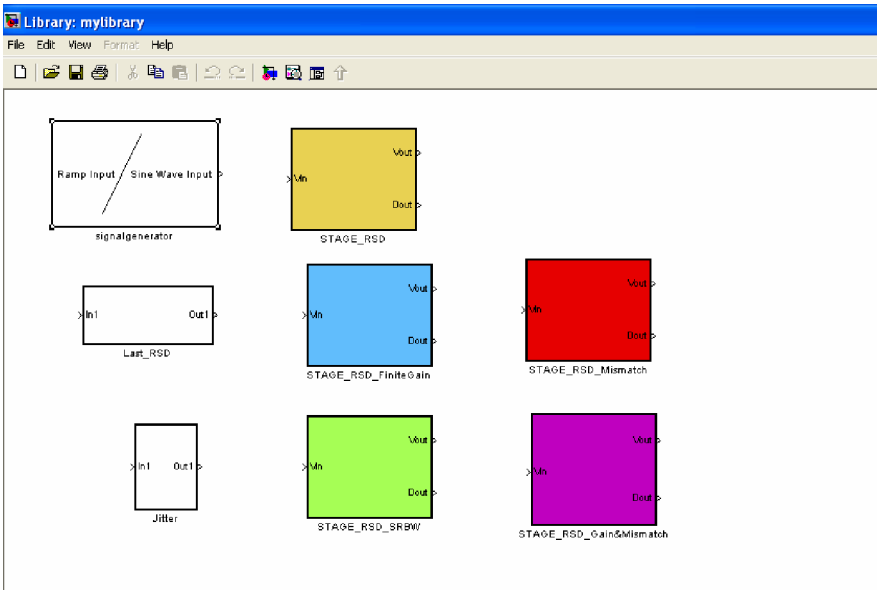


Figura 7.1.: Librería mylibrary.mdl

Llegados a este punto, es interesante ver cómo se ha realizado cada etapa:

- STAGE_RSD: Es la etapa ideal (sólo consideraría el offset del DAC en caso de que le diésemos un valor distinto de 0), podemos ver su estructura en la figura 7.2., donde se aprecia claramente que la ganancia del amplificador de residuo es de 2.

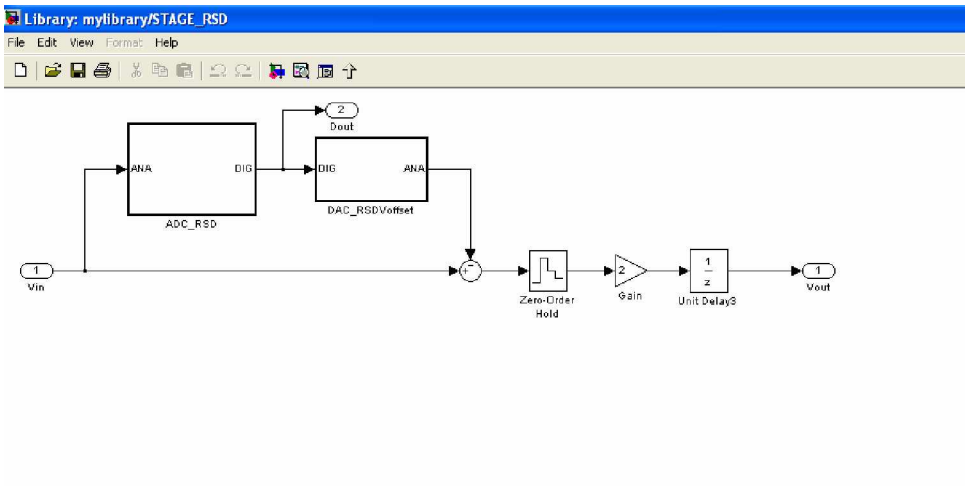


Figura 7.2.: Bloque STAGE_RSD

- STAGE_RSD_FiniteGain: Esta etapa considera la ganancia finita del opamp y el offset del DAC. Podemos ver su estructura en la figura 7.3., donde, concordando con la ecuación (3.2) que se dedujo en el capítulo 3, la ganancia del amplificador de residuo es de $K = \frac{2}{1+\frac{2}{A_0}}$ (con A_0 la ganancia finita del opamp en unidades naturales).

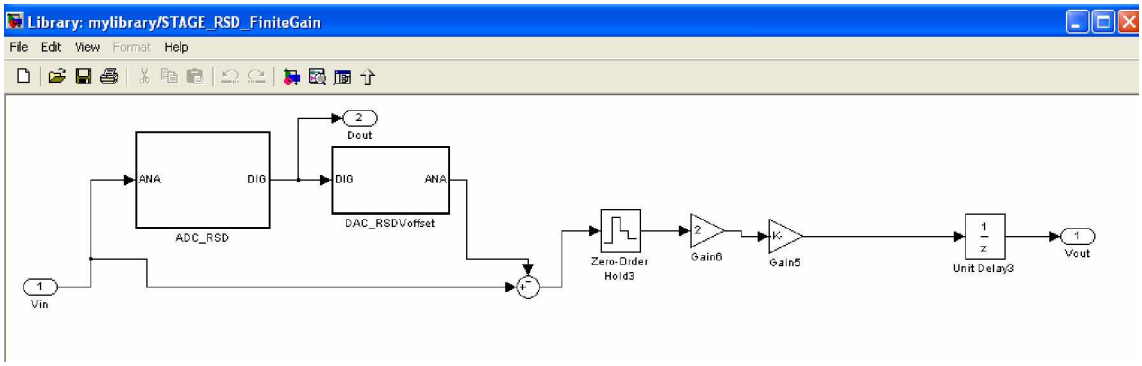


Figura 7.3.: Bloque STAGE_RSD_FiniteGain

- STAGE_RSD_Mismatch: Esta etapa considera el desapareamiento entre las capacidades y el offset del DAC. Podemos ver su estructura en la figura 7.4., donde, concordando con la ecuación (3.7) que se dedujo en el capítulo 3, se ha introducido una ganancia de $K = \frac{Eps}{100}$ (con Eps el desapareamiento entre las capacidades en tanto por uno).

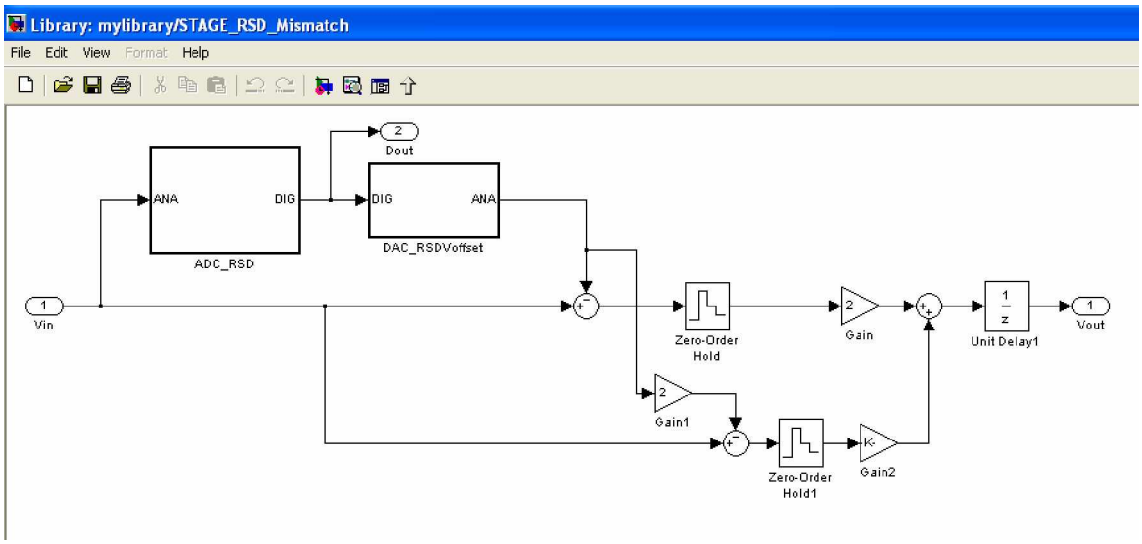


Figura 7.4.: Bloque STAGE_RSD_Mismatch

- STAGE_RSD_Gain&Mismatch: Esta etapa agrupa a las 2 anteriores (considera ganancia finita, desapareamiento y offset). Podemos ver su estructura en la figura 7.5., donde se han englobado las ecuaciones (3.2) y (3.7), es decir, las figuras 7.3. y 7.4.

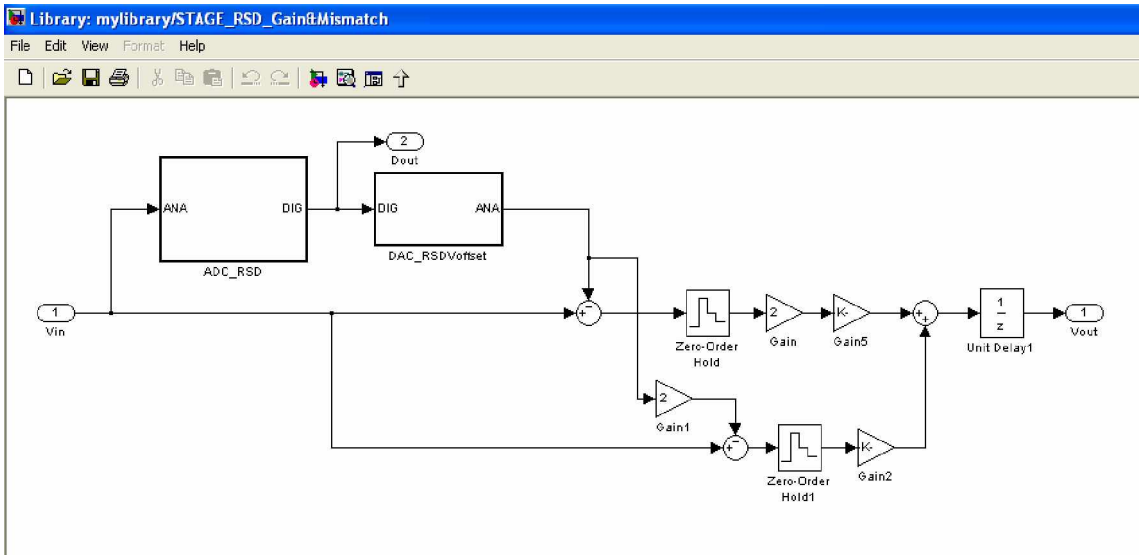


Figura 7.5.: Bloque STAGE_RSD_Gain&Mismatch

- STAGE_RSD_SRBW: Esta etapa considera la no idealidad producida por el Slew Rate y por el Ancho de Banda, y también el offset del DAC. Podemos ver su estructura en la figura 7.6., donde se ha implementado la ecuación (3.16) mediante la subetapa SR-BWopamp.

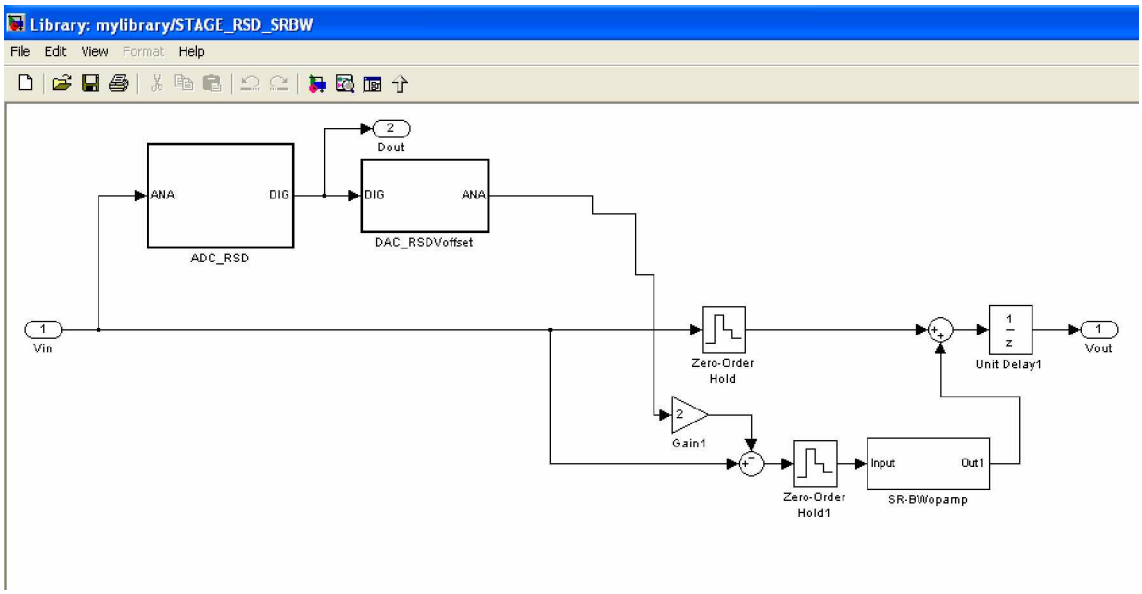


Figura 7.6.: Bloque STAGE_RSD_SRBW

7.2.2. Substages.mdl

El aspecto de esta librería puede verse en la figura 7.7. Se aprecia que tiene únicamente tres bloques: ADC, DAC con no idealidad de offset, y función para generar no idealidad de Slew Rate y Ancho de Banda.

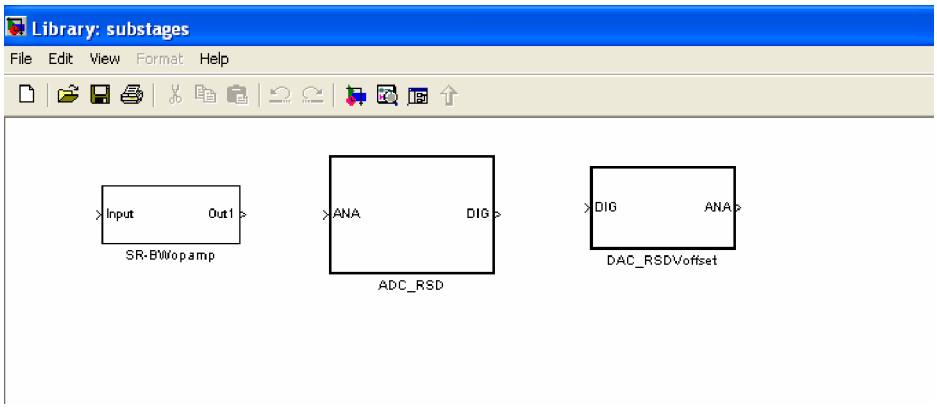


Figura 7.7.: Librería substages.mdl

7.2.3. Digcorrect.mdl

El aspecto de esta librería puede verse en la figura 7.8. Se aprecia que contiene todos los bloques posibles, en el rango de resolución del pipeline de 3 a 15 bits, para la corrección digital del convertidor.

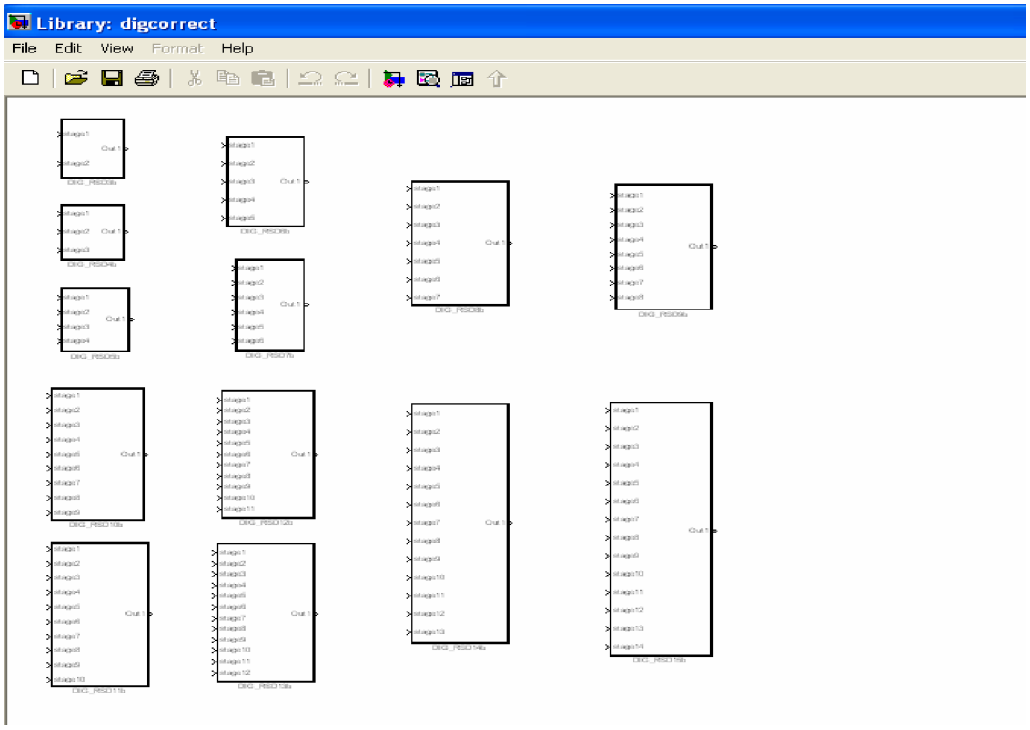


Figura 7.8.: Librería digcorrect.mdl

Puede observarse el aspecto de uno de estos bloques (en este caso, mostraremos el bloque de corrección digital de 10 bits) en la figura 7.9., donde la ganancia K de Gain7 es de 128, y la ganancia K de Gain8 es de 256.

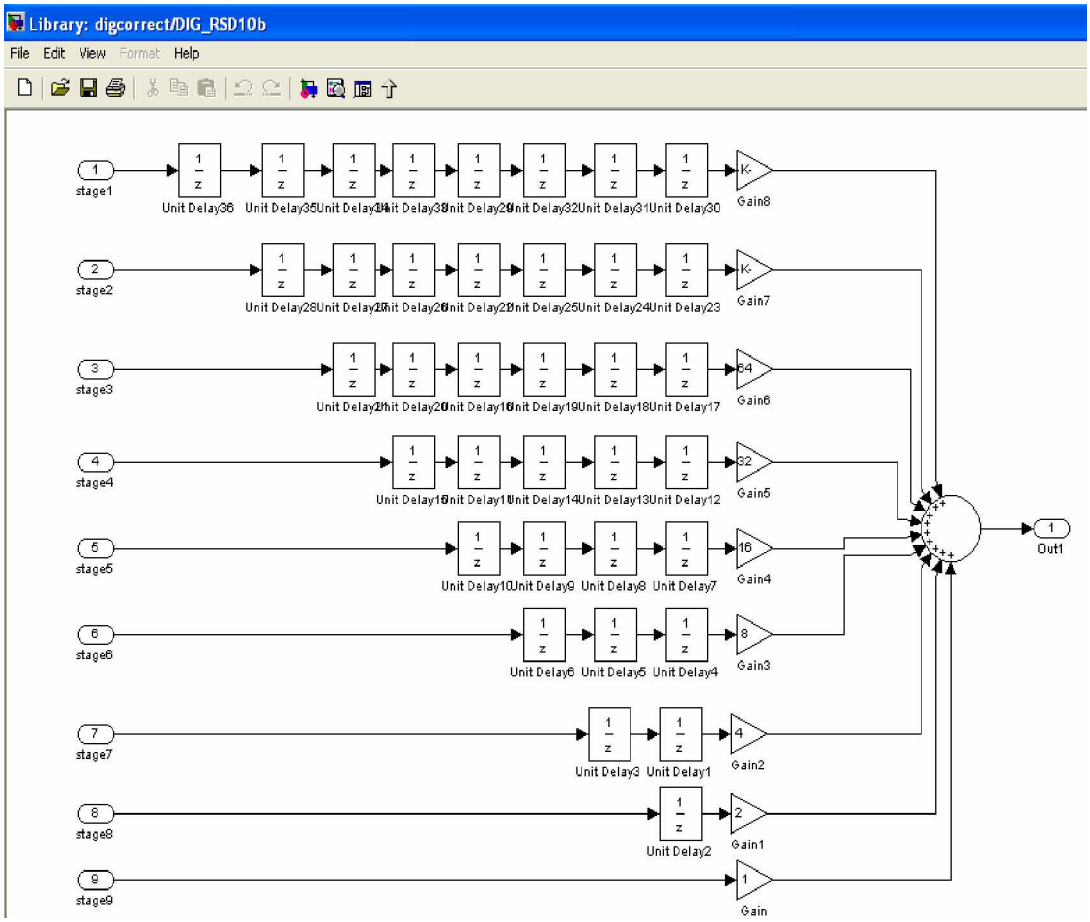


Figura 7.9.: Bloque de corrección digital de 10 bits

7.2.4. Parts6bits.mdl

Esta librería contiene todos los elementos necesarios (los describiremos a continuación) para la construcción de distintos tipos de modelos para convertidores flash de 6 bits:

- GenVref6bits: Bloque que genera las tensiones de referencia para los comparadores.
- Comparator: Bloque que compara un valor con otro.
- Comparator Ts: Igual que Comparator pero muestrea una de las entradas con tiempo de muestreo Ts.

- Comparator Ts Offset: Igual que Comparator Ts pero además incluye el efecto del offset del comparador.

- Comparator Ts Offset Relay: Igual que Comparator Ts Offset, incluyendo el efecto de la histéresis del comparador.

- Comparator Ts Offset Relay Pole: Igual que Comparator Ts Offset Relay, pero incluyendo el efecto de un polo.

- Comparator Bank 6bits: Banco de 63 comparadores de tipo Comparator, con un tiempo de muestreo común.

- Comparator Ts Bank 6bits: Banco de 63 comparadores de tipo Comparator Ts, con un tiempo de muestreo configurable para cada comparador.

- Comparator Ts Offset Bank 6bits: Banco de 63 comparadores de tipo Comparator Ts Offset, con un tiempo de muestreo y un offset configurables para cada comparador.

- Comparator Ts Offset Relay Bank 6bits: Banco de 63 comparadores de tipo Comparator Ts Offset Relay, con un tiempo de muestreo, un offset y una histéresis configurables para cada comparador.

- Comparator Ts Offset Relay Pole Bank 6bits: Banco de 63 comparadores de tipo Comparator Ts Offset Relay Pole, con un tiempo de muestreo, un offset, una histéresis y un polo configurables para cada comparador.

- DigToThermo: Bloque que convierte las salidas digitales del banco de comparadores a código termométrico.

- Binary Logic Converter: Bloque que convierte de código termométrico a código binario.

- ConvBitstoNumber: Bloque que recibe como entrada un valor binario de 6 bits y lo convierte en un número decimal.

7.3 Modelos

Se han puesto a la disposición de los usuarios los siguientes modelos:

- Current.mdl: Este modelo es el molde que proporcionamos al usuario para que éste cree sus modelos, porque sólo contiene un generador de señal y un Scope para visualizar, quedando el resto vacío y pudiendo introducirse ahí el modelo que se desea simular. El aspecto de este modelo puede verse en la figura 7.10.

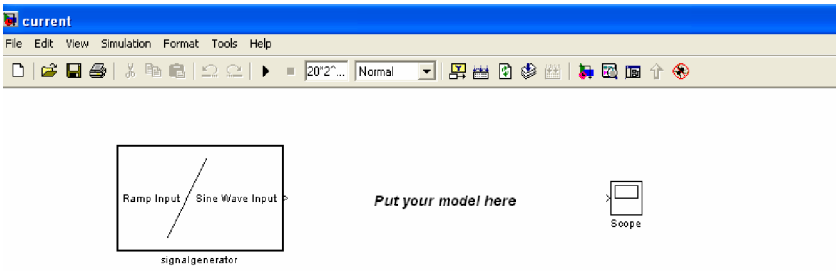


Figura 7.10.: Modelo current.mdl

- Simulacionnoideal10bitsFiniteGain.mdl: En este modelo se representa un pipeline de 10 bits, considerándose las siguientes no idealidades: jitter, offset de los DACs, y ganancia finita (sólo de las dos primeras etapas). Su aspecto puede verse en la figura 7.11.

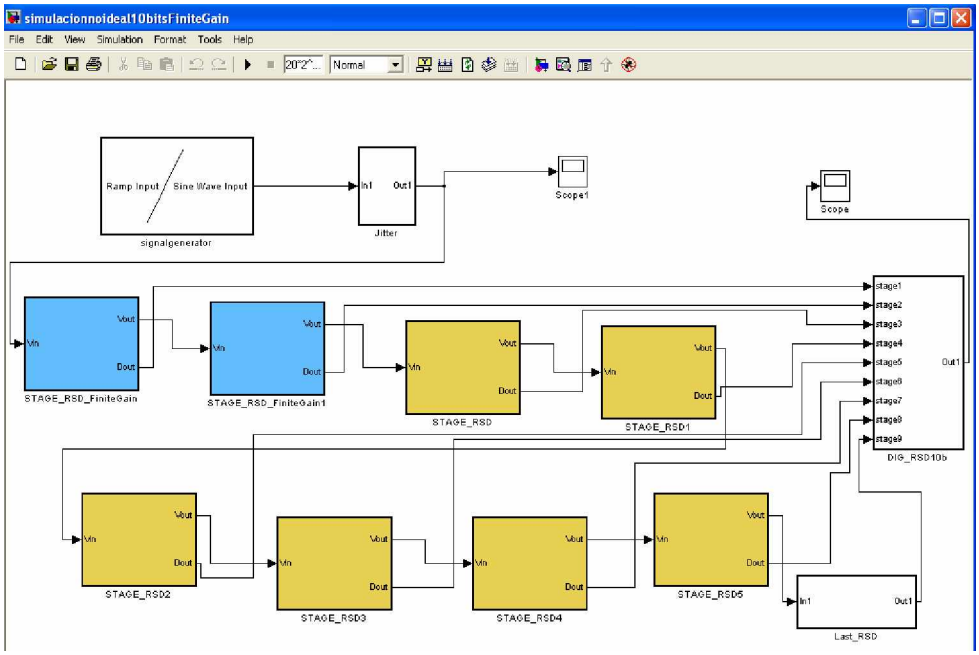


Figura 7.11.: Modelo simulacionnoideal10bitsFiniteGain.mdl

- Simulacionnoideal10bitsMismatch.mdl: En este modelo se representa un pipeline de 10 bits, considerándose las siguientes no idealidades: jitter, offset de los DACs, y desapareamiento de las capacidades (sólo de la primera etapa). Su aspecto puede verse en la figura 7.12.

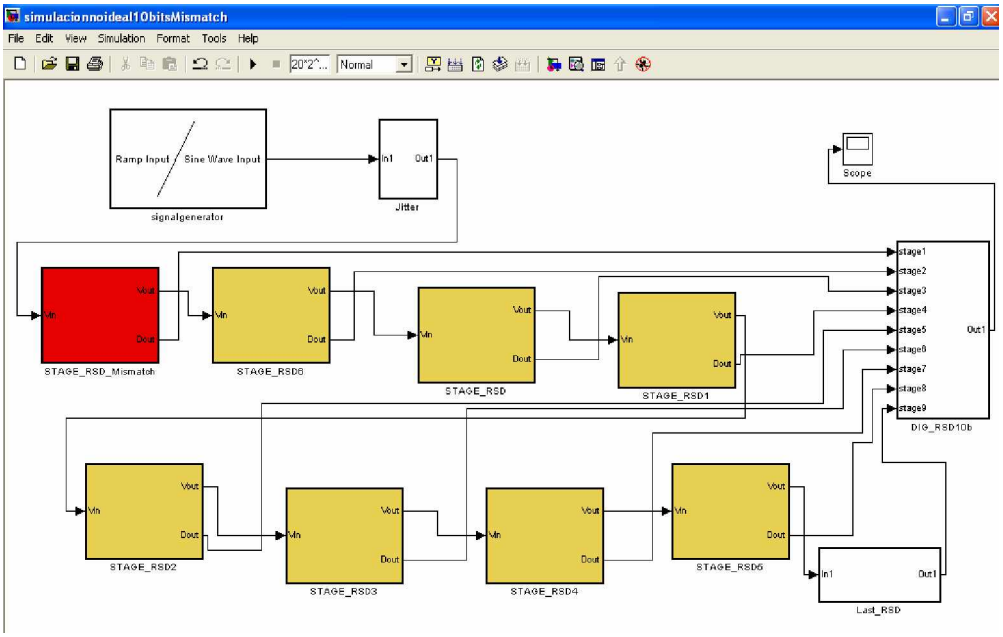


Figura 7.12.: Modelo simulacionnoideal10bitsMismatch.mdl

- Simulacionnoideal10bitsFiniteGainandMismatch.mdl: En este modelo se representa un pipeline de 10 bits, considerándose las siguientes no idealidades: jitter, offset de los DACs, y ganancia finita y desapareamiento de las capacidades (sólo de la primera etapa). Su aspecto puede verse en la figura 7.13.

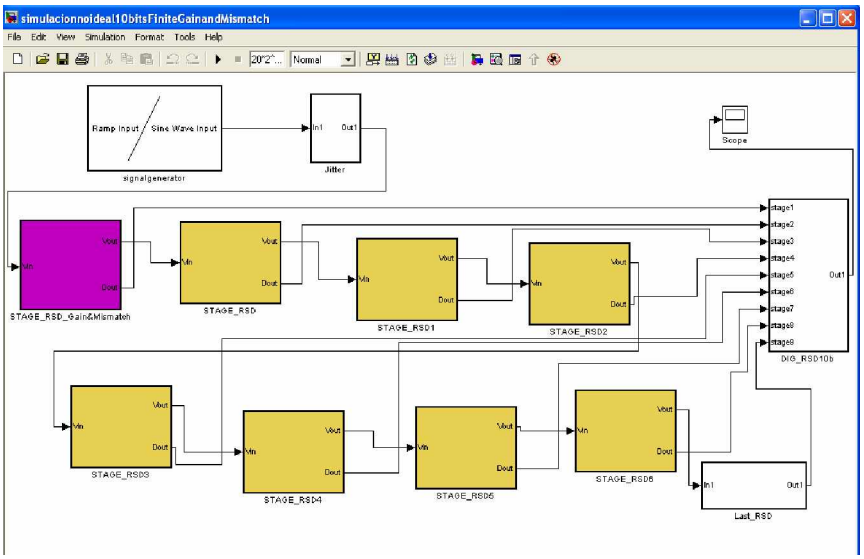


Figura 7.13.: Modelo simulacionnoideal10bitsFiniteGainandMismatch.mdl

- Simulacionnoideal10bitsSR.mdl: En este modelo se representa un pipeline de 10 bits, considerándose las siguientes no idealidades: jitter, offset de los DACs, y Slew Rate y Ancho de Banda. Su aspecto puede verse en la figura 7.14.

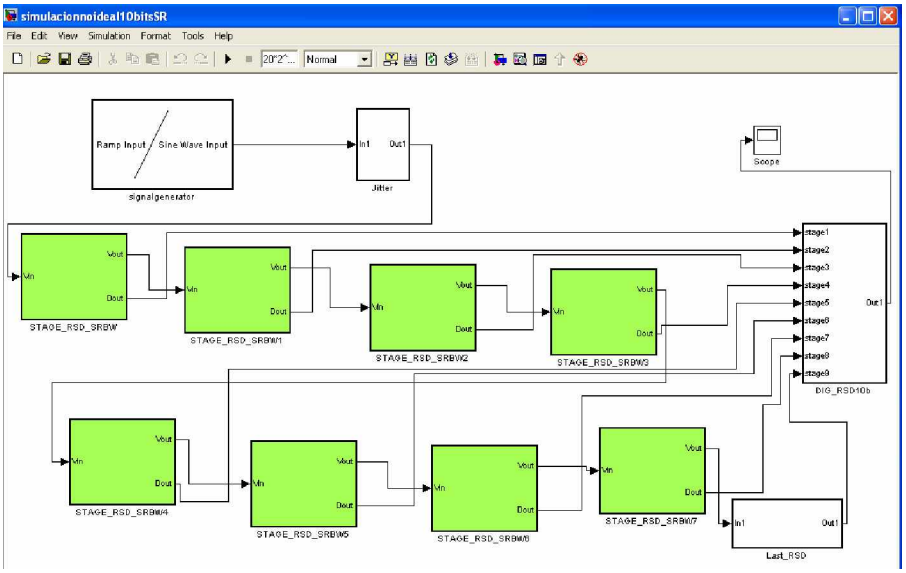


Figura 7.14.: Modelo simulacionnoideal10bitsSR.mdl

- Flashwithoffsetandrelay.mdl: En este modelo se representa un convertidor flash de 6 bits, considerándose las siguientes no idealidades: offset en las tensiones de referencia, offset en los comparadores, histéresis en los comparadores. Contiene los bloques GenVref6bits, Comparator Ts Offset Relay Bank 6bits, DigToThermo, Binary Logic Converter, y ConvBitstoNumber. Su aspecto puede verse en la figura 7.15.

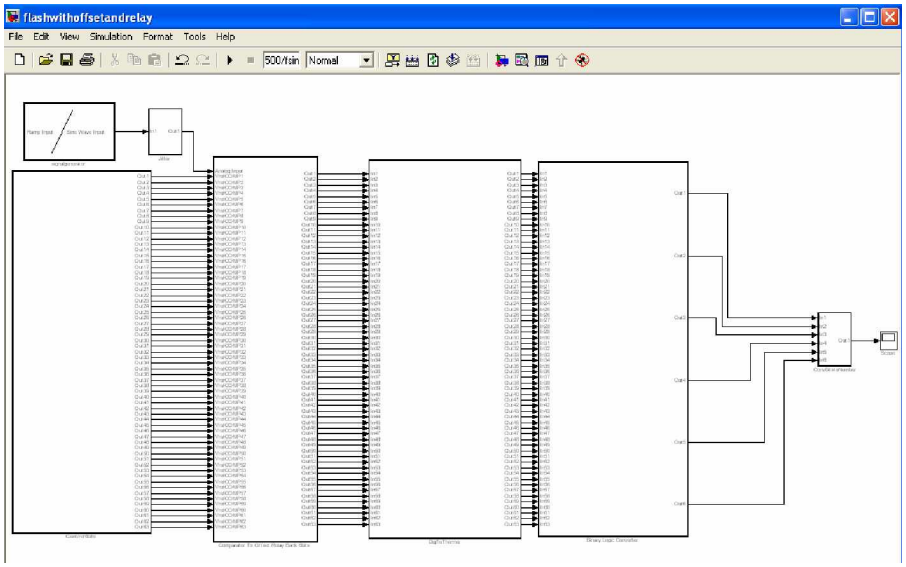


Figura 7.15.: Modelo flashwithoffsetandrelay.mdl

Capítulo 8
Resultados de las simulaciones en Matlab

8.1 Introducción

En el presente capítulo se expondrán los resultados obtenidos de la simulación de diversos convertidores con la aplicación. En concreto, se estudiará el convertidor pipeline de 10 bits y el convertidor flash de 6 bits.

Debido a que este proyecto no tenía como objetivo la realización de un estudio exhaustivo de los convertidores reseñados, se mostrarán los resultados más significativos pero no de forma pormenorizada.

Con el programa implementado, se pueden realizar 4 tipos principales de simulaciones: simulaciones ideales (son simulaciones simples sin tener en cuenta las no idealidades), simulaciones no ideales (simulaciones simples teniendo en cuenta las no idealidades), barridos simples (múltiples simulaciones variando un parámetro) y barridos dobles (múltiples simulaciones variando 2 parámetros). Teniendo en cuenta que el objetivo no era hacer un estudio exhaustivo, se ha decidido omitir las simulaciones no ideales, ya que con los barridos es suficiente para hacerse una idea aproximada de cómo afectan las variaciones de parámetros al convertidor.

Pasamos ya a los resultados propiamente dichos para el pipeline de 10 bits y el flash de 6 bits. Dedicaremos a ello los dos próximos apartados.

8.2 Convertidor pipeline de 10 bits

Comenzaremos con las simulaciones ideales de este convertidor, después pasaremos a los barridos simples y por último expondremos los barridos dobles:

- **Pipeline Simulación Nº 1:**

Datos de la simulación:

Tipo: Simulación ideal
Entrada: Rampa
Modelo: Simulacionnoideal10bitsFiniteGain.mdl

El aspecto de la GUI puede verse en la figura 8.1., en la que pueden apreciarse los valores de las variables de entrada (zona de Inputs) y de las no idealidades (zona de Non Idealities), y los resultados siguientes (zona de Results):

MaxINL=0.03751
MaxDNL=0.050047
Missing Codes=0

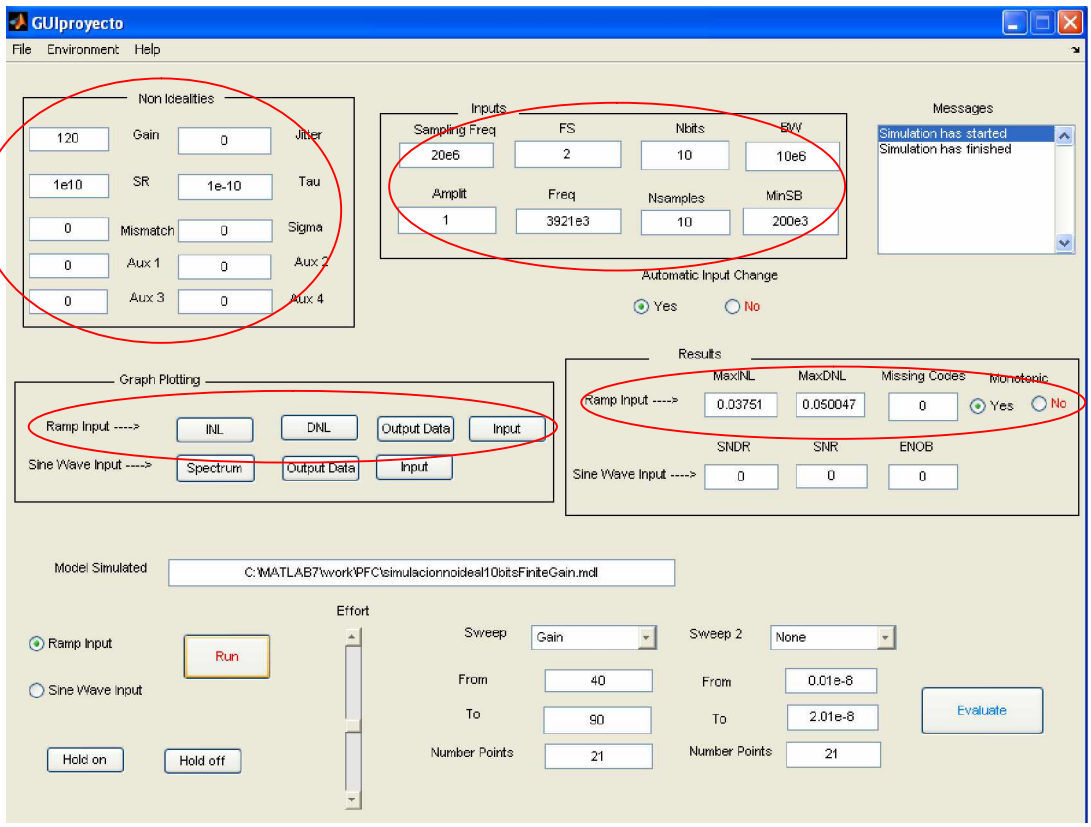


Figura 8.1.: GUI tras Pipeline Simulación Nº 1

Las gráficas de INL, DNL, Output Data e Input pueden verse en las figuras 8.2., 8.3., 8.4. y 8.5. respectivamente. Puede apreciarse que el valor

máximo del valor absoluto de la gráfica de INL coincidirá con el resultado MaxINL (MaxINL tiene en cuenta los códigos perdidos pero, al salir los códigos perdidos 0, coincide el máximo del valor absoluto de la gráfica con MaxINL). De la misma forma, coincide el máximo del valor absoluto de la gráfica de DNL con MaxDNL (en MaxDNL no se tienen en cuenta los códigos perdidos, luego el máximo de la gráfica de DNL siempre coincidirá con MaxDNL).

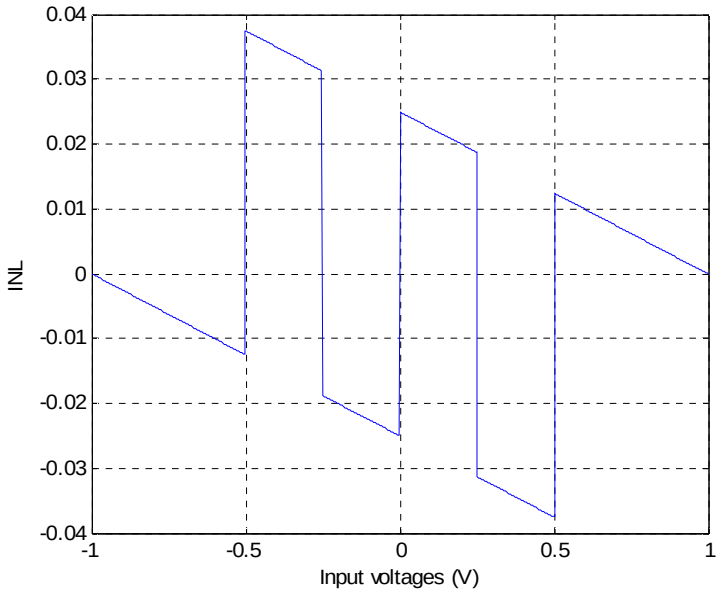


Figura 8.2.: Gráfica INL Pipeline Simulación Nº 1

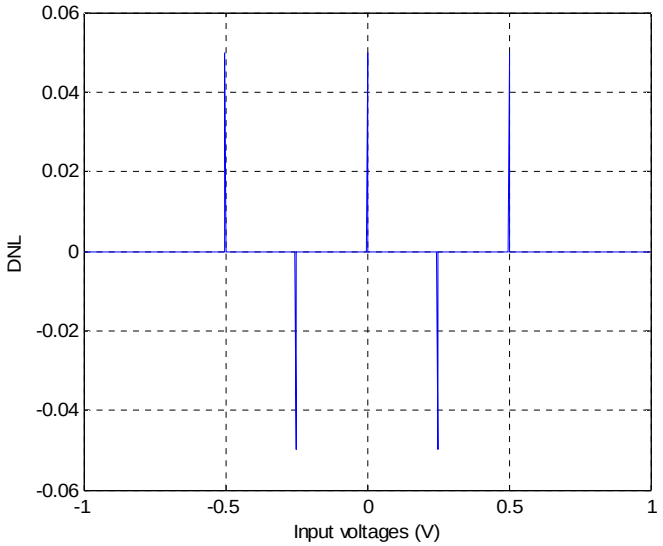


Figura 8.3.: Gráfica DNL Pipeline Simulación Nº 1

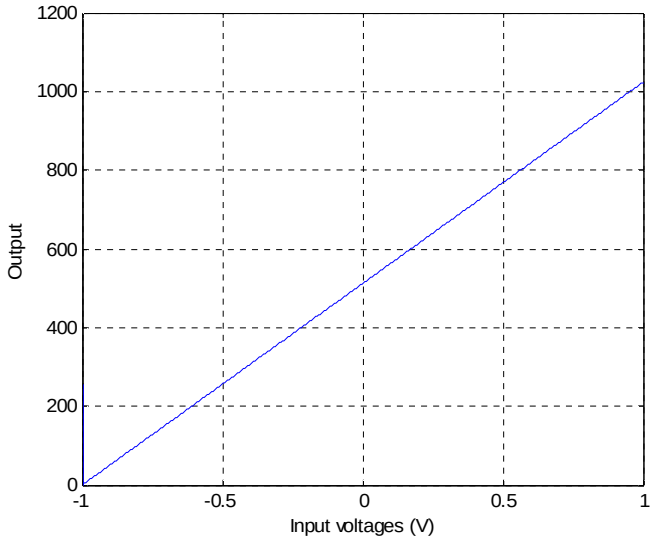


Figura 8.4.: Gráfica Output Data Pipeline Simulación Nº 1

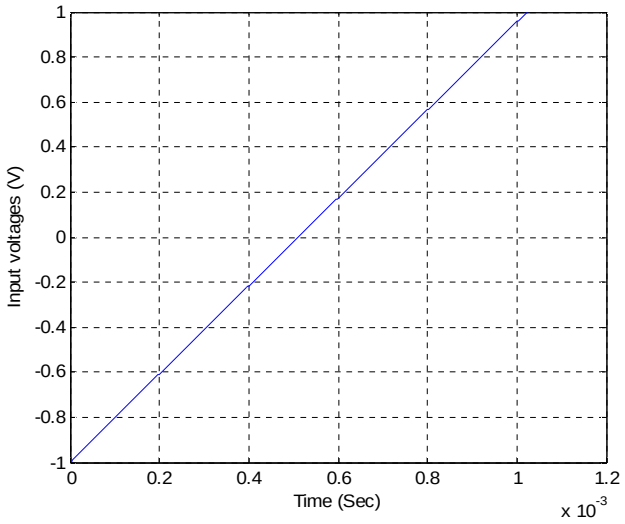


Figura 8.5.: Gráfica Input Pipeline Simulación Nº 1

- **Pipeline Simulación Nº 2:**

Datos de la simulación:

Tipo: Simulación ideal
Entrada: Rampa
Modelo: Simulacionnoideal10bitsMismatch.mdl

El aspecto de la GUI es similar al de la figura 8.1., ya que se han elegido los mismos valores de las variables de entrada y de las no idealidades, sólo hemos cambiado el modelo. Se obtienen los resultados siguientes:

MaxINL=0.05
MaxDNL=0.05
Missing Codes=0

Las gráficas de INL y DNL pueden verse en las figuras 8.6. y 8.7. respectivamente. Las gráficas de Output Data e Input resultantes son idénticas a las obtenidas en la Pipeline Simulación Nº 1 (figuras 8.4. y 8.5.). Todos los resultados de esta simulación son idénticos a los que se obtendrían con el modelo Simulacionnoideal10bitsSR.mdl.

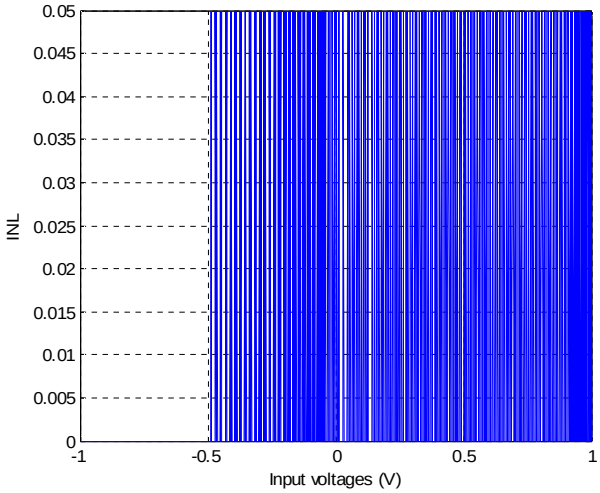


Figura 8.6.: Gráfica INL Pipeline Simulación Nº 2

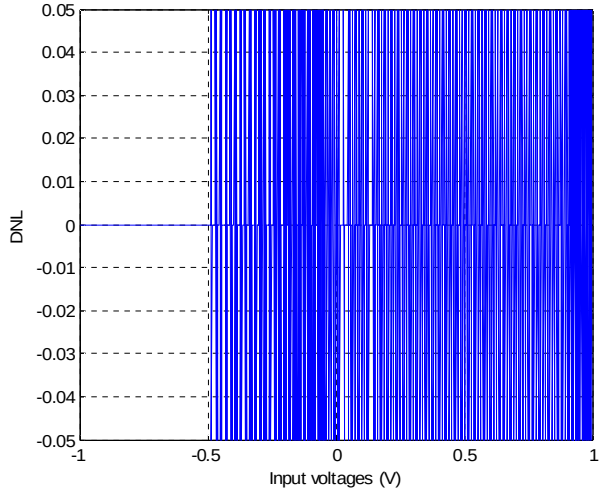


Figura 8.7.: Gráfica DNL Pipeline Simulación Nº 2

- **Pipeline Simulación Nº 3:**

Datos de la simulación:

Tipo: Simulación ideal
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl

El aspecto de la GUI puede verse en la figura 8.8., en la que pueden apreciarse los valores de las variables de entrada (zona de Inputs) y de las no idealidades (zona de Non Idealities), y los resultados siguientes (zona de Results):

SNDR=61.9508
SNR=62.0351
ENOB=9.9985

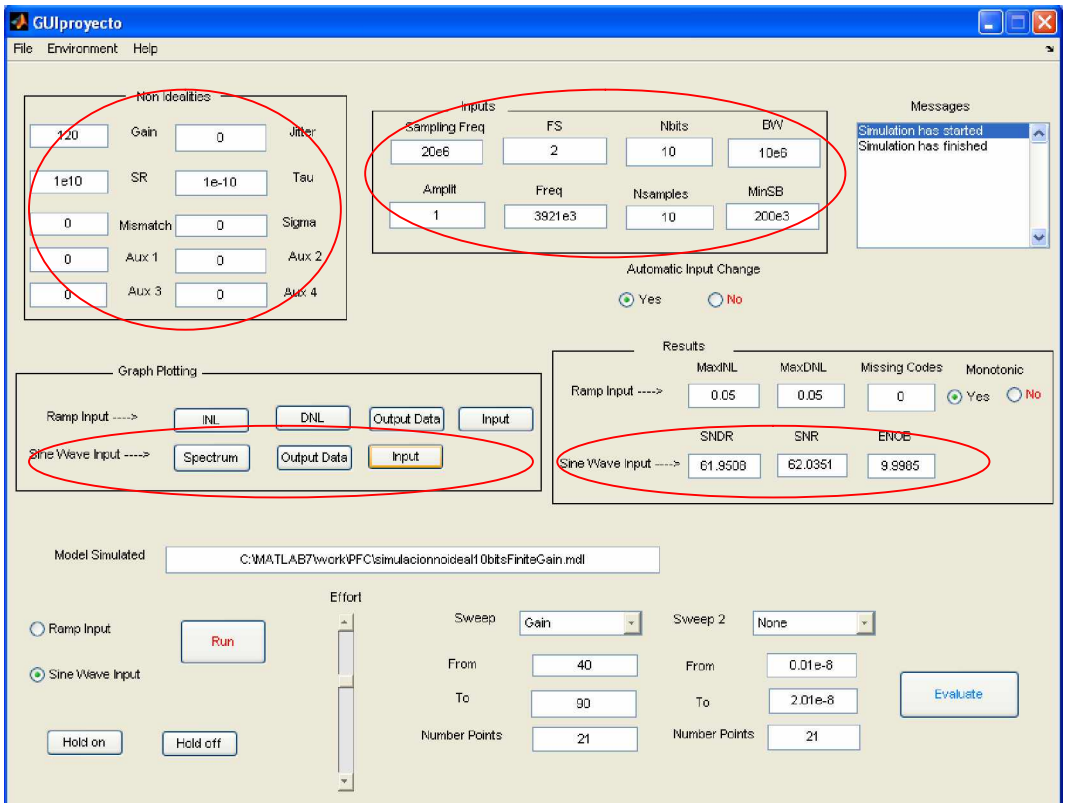


Figura 8.8.: GUI tras Pipeline Simulación Nº 3

Las gráficas de Spectrum, Output Data e Input pueden verse en las figuras 8.9., 8.10. y 8.11. respectivamente. Todos los resultados de esta simulación son idénticos a los que se obtendrían con los modelos Simulacionnoideal10bitsMismatch.mdl y Simulacionnoideal10bitsSR.mdl.

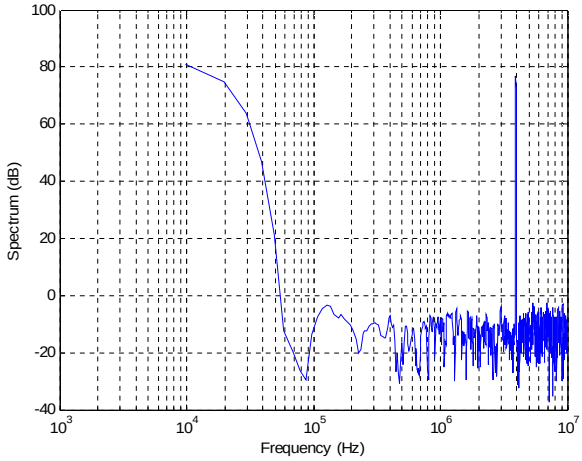


Figura 8.9.: Gráfica Spectrum Pipeline Simulación Nº 3

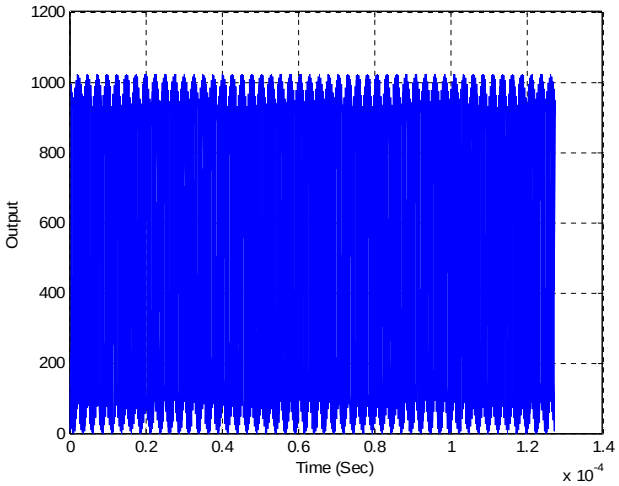


Figura 8.10.: Gráfica Output Data Pipeline Simulación Nº 3

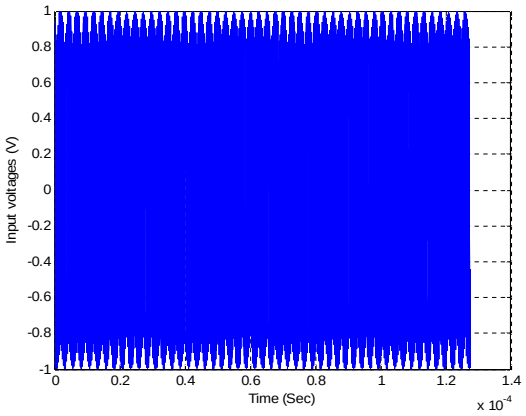


Figura 8.11.: Gráfica Input Pipeline Simulación Nº 3

- **Pipeline Simulación Nº 4:**

Datos de la simulación:

Tipo: Barrido simple de Ganancia
Entrada: Rampa
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 5
Start: 40
Stop: 90
Points: 21

La gráfica resultante puede verse en la figura 8.12., y una ampliación en la zona de mayor interés en la figura 8.13., de ésta última se extrae que, para asegurar que INL sea menor de 0.25, la ganancia del amplificador utilizado para las dos primeras etapas debe ser superior a 70dB.

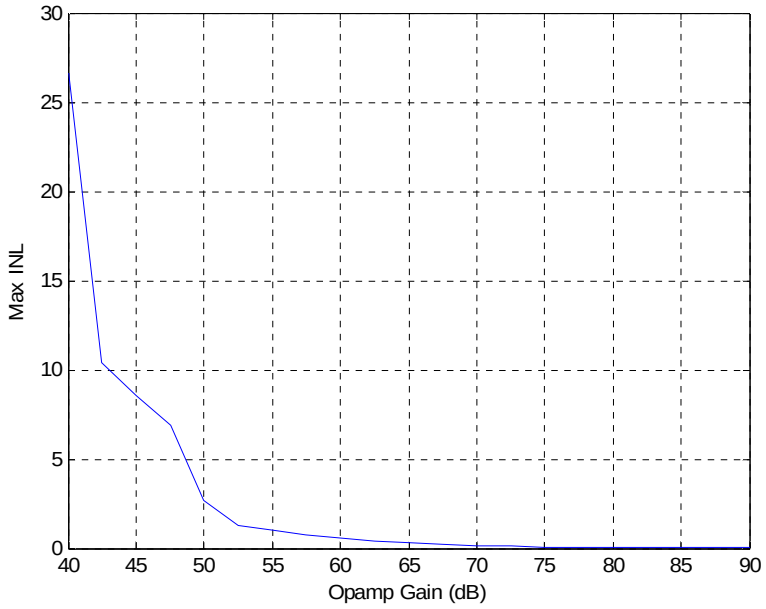


Figura 8.12.: Gráfica de MaxINL frente a la ganancia DC del amplificador de las dos primeras etapas

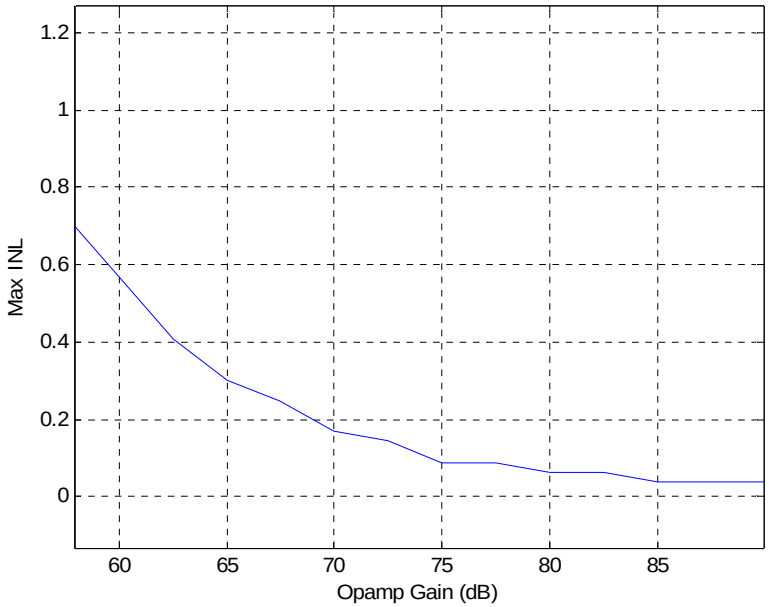


Figura 8.13.: Ampliación de la figura 8.12 en la zona de mayor interés

- **Pipeline Simulación Nº 5:**

Datos de la simulación:

Tipo: Barrido simple de Desapareamiento entre Capacidades
Entrada: Rampa
Modelo: Simulacionnoideal10bitsMismatch.mdl
Effort: 5
Start: -0.2
Stop: 0.2
Points: 41

La gráfica resultante puede verse en la figura 8.14., y una ampliación en la zona de mayor interés en la figura 8.15., de ésta última se extrae que, para asegurar que INL sea menor de 0.25, el desapareamiento de las capacidades de la primera etapa debe ser inferior al 0.02%.

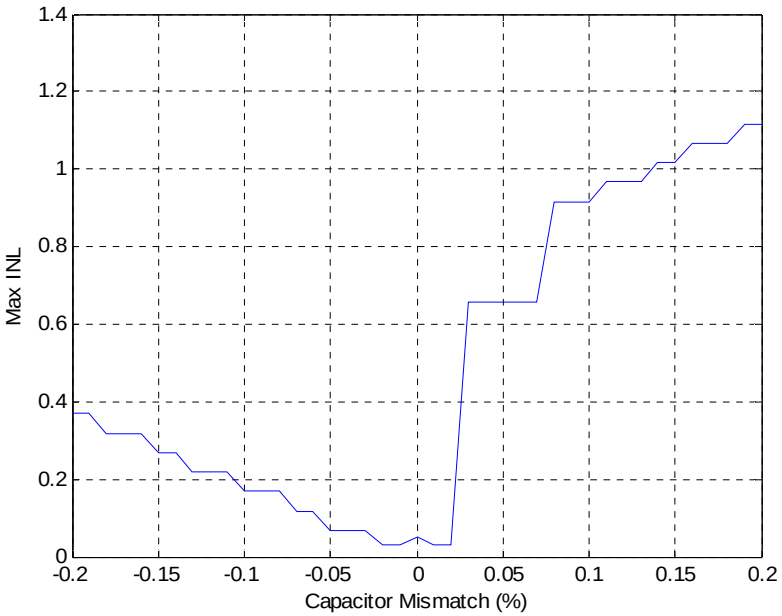


Figura 8.14.: Gráfica de MaxINL frente al desapareamiento de las capacidades de la primera etapa

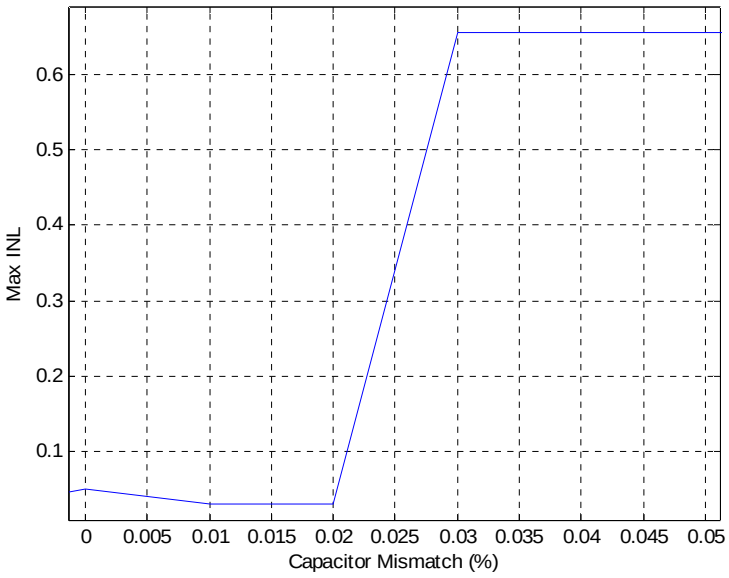


Figura 8.15.: Ampliación de la figura 8.14 en la zona de mayor interés

- **<u>Pipeline Simulación Nº 6:</u>**

Datos de la simulación:

Tipo: Barrido simple de Slew Rate
Entrada: Rampa
Modelo: Simulacionnoideal10bitsSR.mdl
Effort: 5
Start: 1e7
Stop: 2e7
Points: 11

La gráfica resultante puede verse en la figura 8.16., y una ampliación en la zona de mayor interés en la figura 8.17., de ésta última se extrae que, para asegurar que INL sea menor de 0.25, el Slew Rate debe ser mayor que 1.5e7.

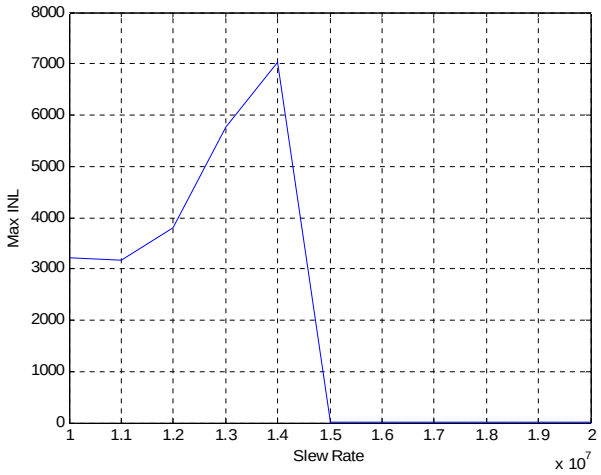


Figura 8.16.: Gráfica de MaxINL frente al Slew Rate

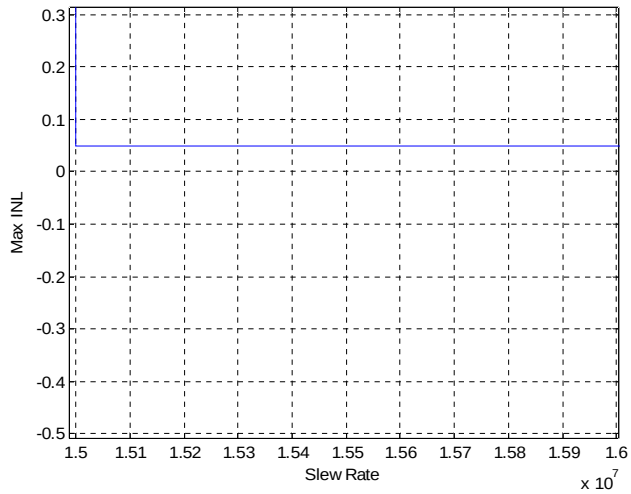


Figura 8.17.: Ampliación de la figura 8.15 en la zona de mayor interés

- **Pipeline Simulación Nº 7:**

Datos de la simulación:

Tipo: Barrido simple de Tau
Entrada: Rampa
Modelo: Simulacionnoideal10bitsSR.mdl
Effort: 5
Start: 7e-9
Stop: 1.4e-8
Points: 15

La gráfica resultante puede verse en la figura 8.18., y una ampliación en la zona de mayor interés en la figura 8.19., de ésta última se extrae que, para asegurar que INL sea menor de 0.25, 1/(2*pi*Tau) debe ser mayor que 2.1e7, por tanto la constante de tiempo (despejando Tau) deberá ser menor que 7.6 nanosegundos.

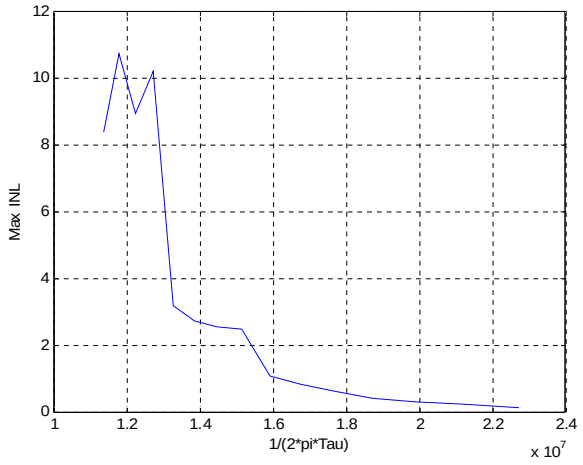


Figura 8.18.: Gráfica de MaxINL frente a 1/(2*pi*Tau)

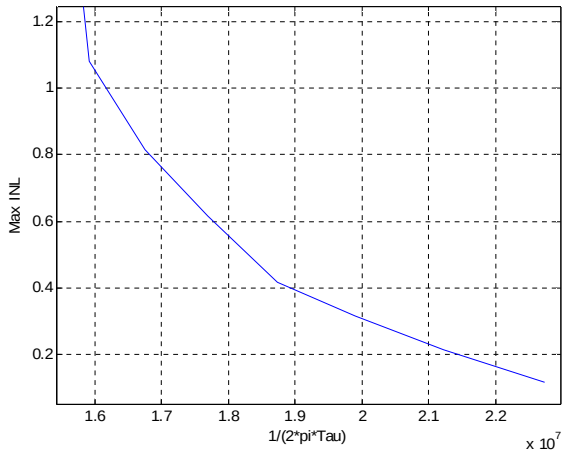


Figura 8.19.: Ampliación de la figura 8.18 en la zona de mayor interés

- **Pipeline Simulación Nº 8:**

Datos de la simulación:

Tipo: Barrido simple de Sigma (offset)
Entrada: Rampa
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 5
Start: 1e-3
Stop: 1e-2
Points: 11

La gráfica resultante puede verse en la figura 8.20., en la que puede apreciarse cómo, a medida que aumentamos la desviación típica del offset, la máxima INL tiende a subir, aunque a veces puede disminuir debido al carácter aleatorio de la variable, pero como hemos dicho en general tiende a subir.

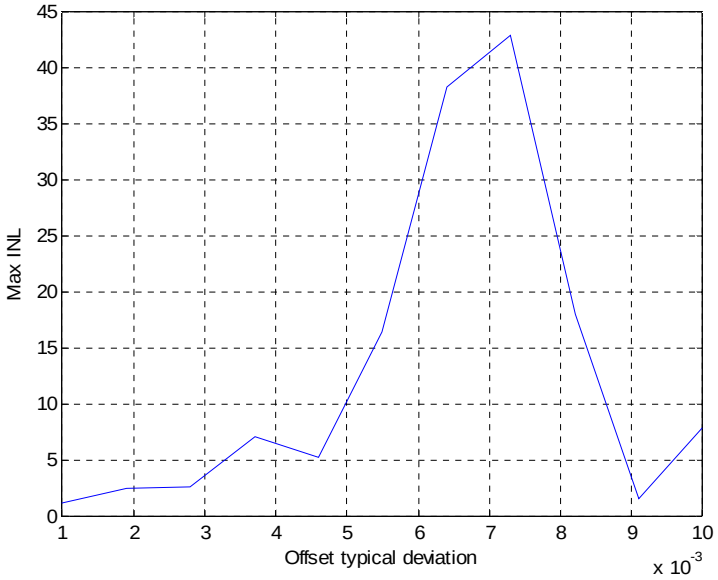


Figura 8.20.: Gráfica de MaxINL frente a la desviación típica del offset

- **Pipeline Simulación Nº 9:**

Datos de la simulación:

Tipo: Barrido simple de Ganancia
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 40
Stop: 90
Points: 21

La gráfica resultante puede verse en la figura 8.21., donde puede apreciarse que la caída de 3 dB en la SNDR se sitúa en una ganancia entre los 55 y los 60 dB.

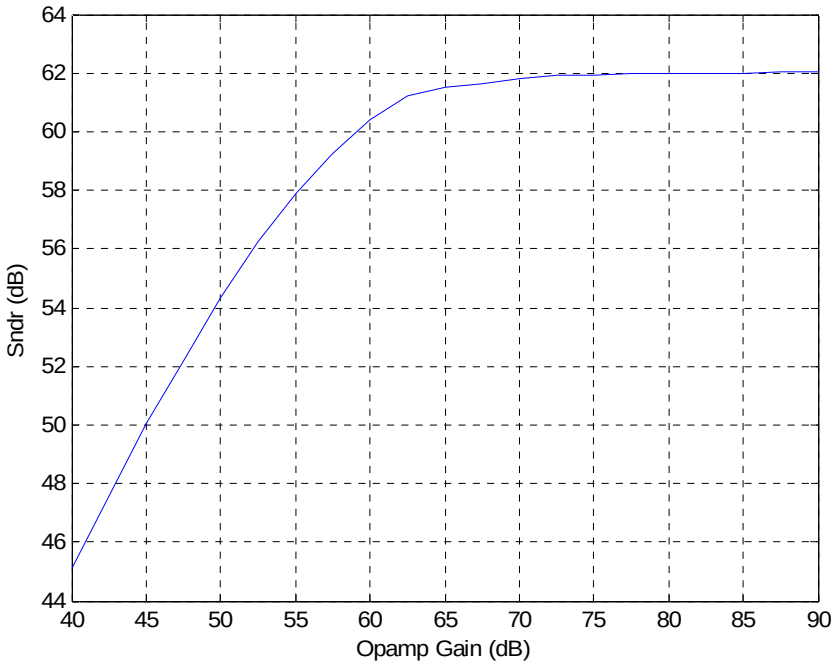


Figura 8.21.: Gráfica de SNDR frente a la ganancia DC del amplificador de las dos primeras etapas

- **Pipeline Simulación Nº 10:**

Datos de la simulación:

Tipo: Barrido simple de Desapareamiento entre Capacidades
Entrada: Seno
Modelo: Simulacionnoideal10bitsMismatch.mdl
Effort: 7
Start: -2
Stop: 2
Points: 41

La gráfica resultante puede verse en la figura 8.22., donde puede apreciarse que la caída de 3 dB en la SNDR se sitúa en un desapareamiento de las capacidades algo inferior al 0.5% .

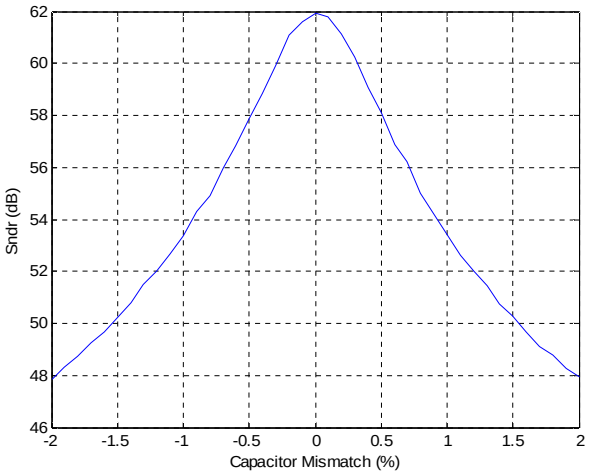


Figura 8.22.: Gráfica de SNDR frente al desapareamiento de las capacidades de la primera etapa

- **Pipeline Simulación Nº 11:**

Datos de la simulación:

Tipo: Barrido simple de Slew Rate
Entrada: Seno
Modelo: Simulacionnoideal10bitsSR.mdl
Effort: 7
Start: 1e7
Stop: 2e7
Points: 11

La gráfica resultante puede verse en la figura 8.23., donde puede apreciarse que la SNDR cae bruscamente cuando el Slew Rate disminuye de 1.5e7 .

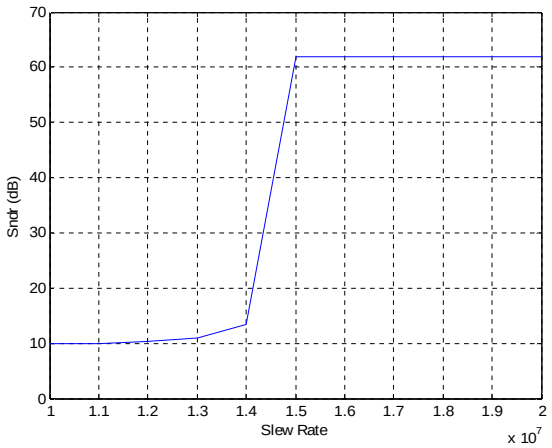


Figura 8.23.: Gráfica de SNDR frente al Slew Rate

- **Pipeline Simulación Nº 12:**

Datos de la simulación:

Tipo: Barrido simple de Tau
Entrada: Seno
Modelo: Simulacionnoideal10bitsSR.mdl
Effort: 7
Start: 7e-9
Stop: 1.4e-8
Points: 15

La gráfica resultante puede verse en la figura 8.24., donde puede apreciarse que la caída de 3 dB en la SNDR se sitúa en un valor de 1/(2*pi*Tau) de entre 1.6e7 y 1.8e7, es decir, entre unos valores de la constante de tiempo (despejando Tau) de entre 9.9 nanosegundos y 8.8 nanosegundos.

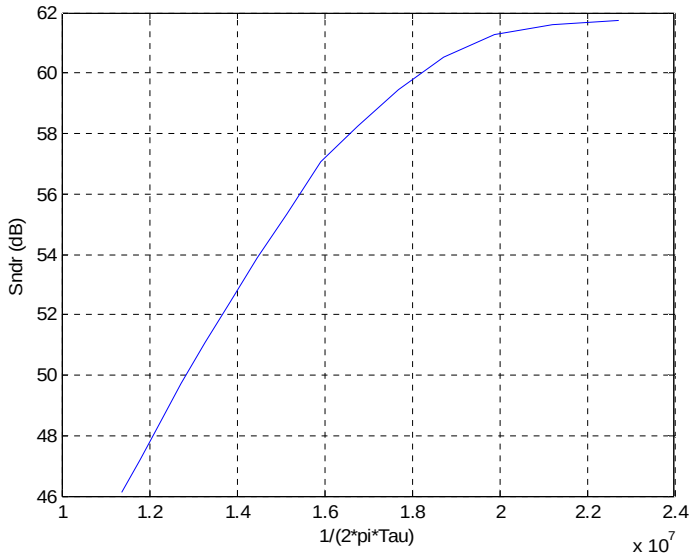


Figura 8.24.: Gráfica de SNDR frente a 1/(2*pi*Tau)

- **Pipeline Simulación Nº 13:**

Datos de la simulación:

Tipo: Barrido simple de Sigma (offset)
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 1e-3
Stop: 1e-2
Points: 11

La gráfica resultante puede verse en la figura 8.25., en la que puede apreciarse cómo, a medida que aumentamos la desviación típica del offset, la

SNDR tiende a disminuir, aunque a veces puede aumentar debido al carácter aleatorio de la variable, pero como hemos dicho en general tiende a disminuir.

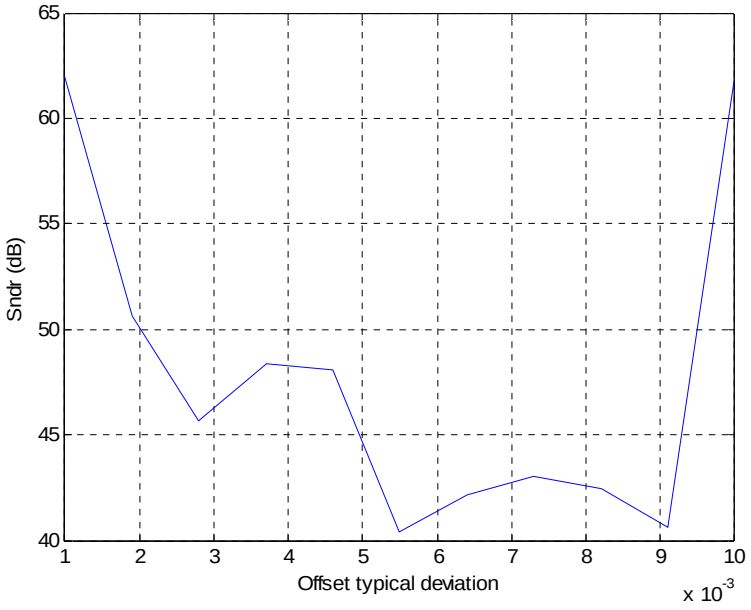


Figura 8.25.: Gráfica de SNDR frente a la desviación típica del offset

- **Pipeline Simulación Nº 14:**

Datos de la simulación:

Tipo: Barrido simple de Jitter
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 1e-10
Stop: 1e-12
Points: 21

La gráfica resultante puede verse en la figura 8.26., donde puede apreciarse que la caída de 3 dB en la SNDR se sitúa en un valor de jitter de entre 300 y 400 picosegundos.

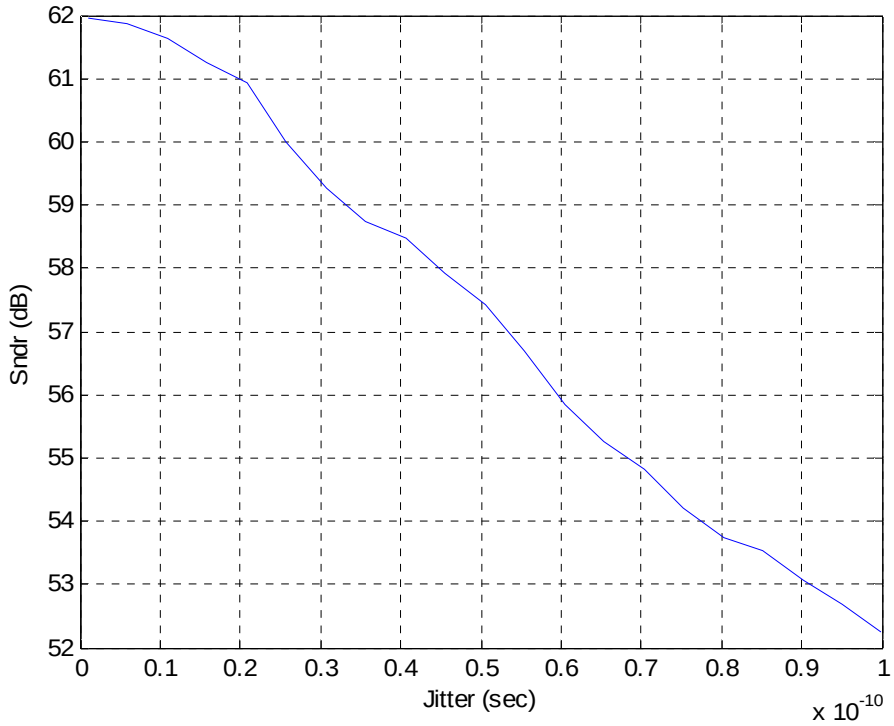


Figura 8.26.: Gráfica de SNDR frente al Jitter

- **Pipeline Simulación Nº 15:**

Datos de la simulación:

Tipo: Barrido simple de Amplitud de entrada
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 0.1
Stop: 1
Points: 21

La gráfica resultante puede verse en la figura 8.27., donde puede apreciarse que la caída de 3 dB en la SNDR se sitúa en un valor de 20*log(2*A/FS) de entre -5dB y -10dB, lo cual, con un fondo de escala de 2 V, supondría una amplitud del seno de entrada de entre 562 mV y 316.2 mV.

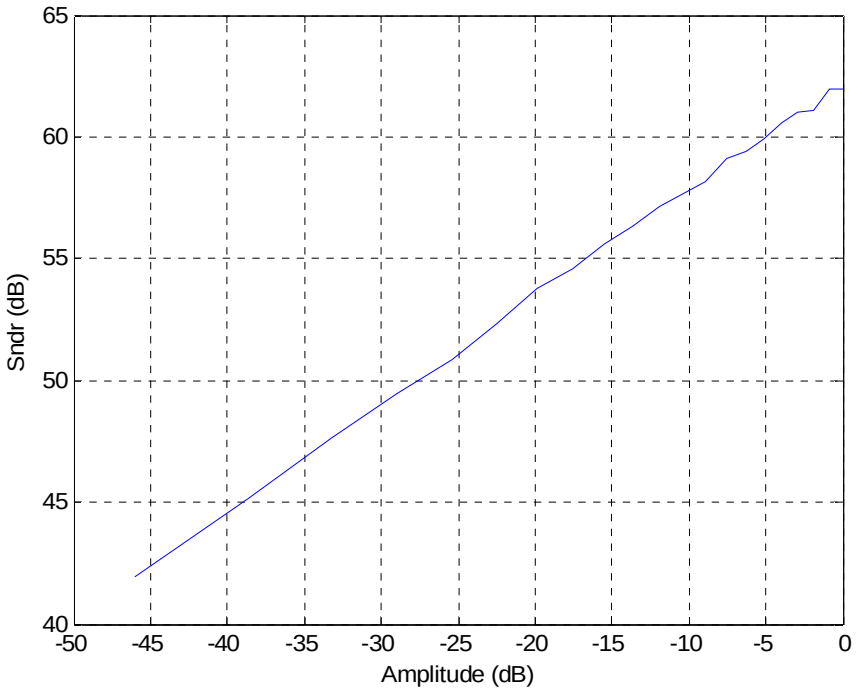


Figura 8.27.: Gráfica de SNDR frente a la Amplitud de entrada

- **Pipeline Simulación Nº 16:**

Datos de la simulación:

Tipo: Barrido simple de Full Scale
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 1
Stop: 3
Points: 21

La gráfica resultante puede verse en la figura 8.28., donde puede apreciarse que, teniendo una amplitud de entrada de 1V, la SNDR cae bruscamente cuando FS disminuye de 2V.

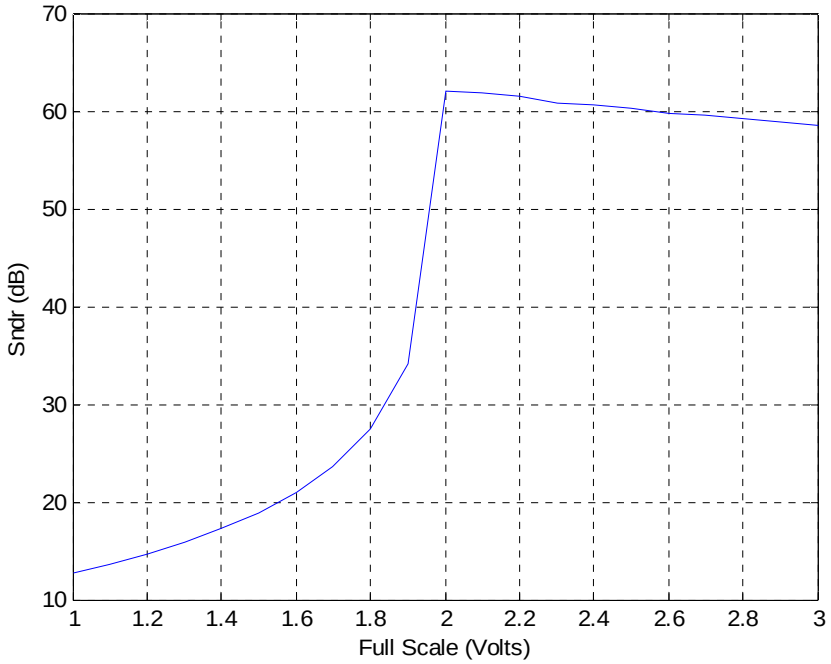


Figura 8.28.: Gráfica de SNDR frente a FS

- **Pipeline Simulación Nº 17:**

Datos de la simulación:

Tipo: Barrido doble de Ganancia y Desapareamiento entre Capacidades
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGainandMismatch.mdl
Effort: 7
Start: 40
Stop: 90
Points: 21
Start2: -0.2
Stop2: 0.2
Points2: 21

La gráfica resultante puede verse en la figura 8.29.:

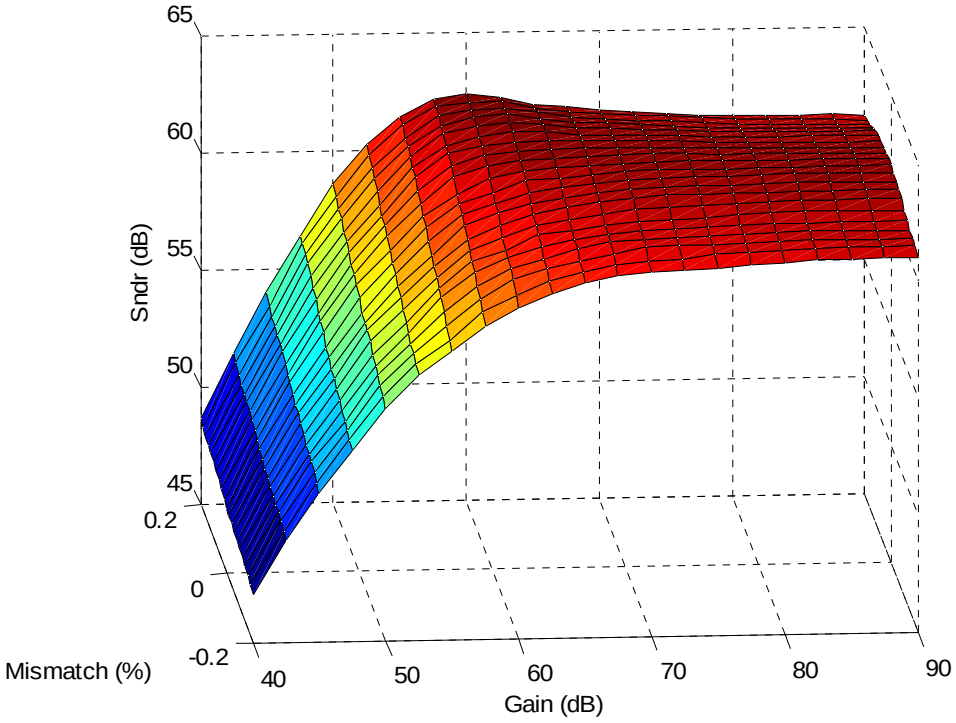


Figura 8.29.: Gráfica de SNDR frente a la Ganancia y al Desapareamiento entre Capacidades

- **Pipeline Simulación Nº 18:**

Datos de la simulación:

Tipo: Barrido doble de Ganancia y Sigma (offset)
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 40
Stop: 90
Points: 21
Start2: 1e-3
Stop2: 1e-2
Points2: 11

La gráfica resultante puede verse en la figura 8.30., donde puede apreciarse que la SNDR va subiendo en la dirección en la que aumenta la ganancia y disminuye la desviación típica del offset, aunque a veces puede disminuir debido al carácter aleatorio del offset, pero como hemos dicho en general tiende a subir en la dirección indicada.

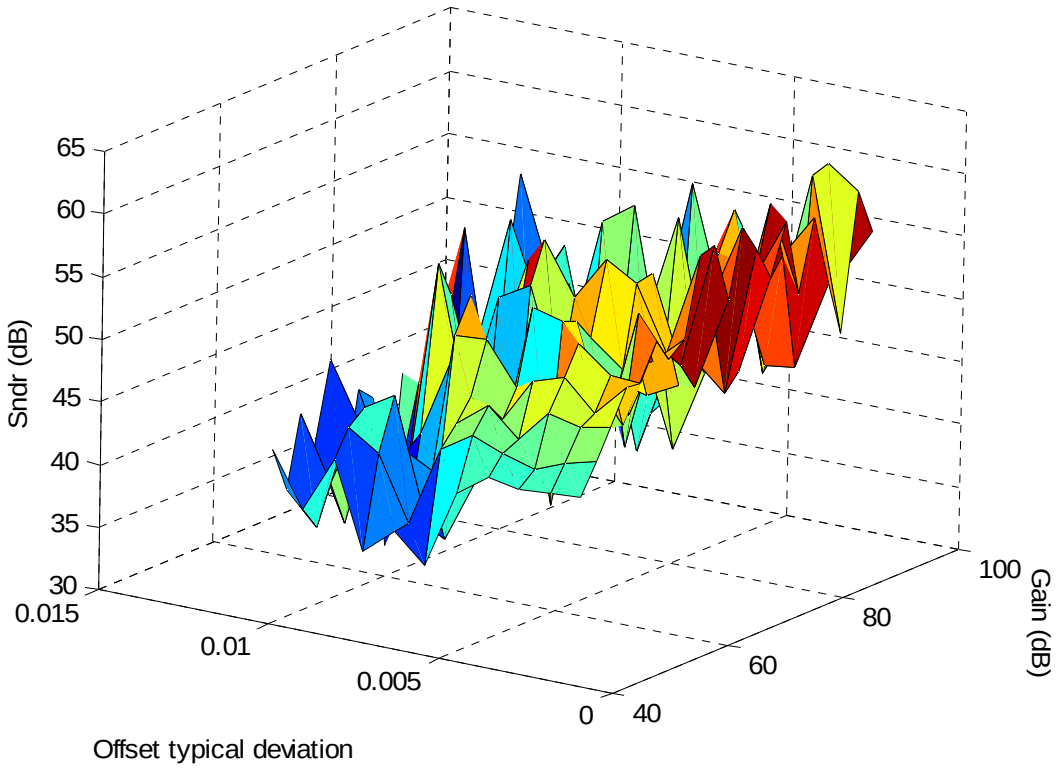


Figura 8.30.: Gráfica de SNDR frente a la Ganancia y a la desviación típica del offset

- **Pipeline Simulación Nº 19:**

Datos de la simulación:

Tipo: Barrido doble de Ganancia y Jitter
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 40
Stop: 90
Points: 21
Start2: 1e-10
Stop2: 1e-12
Points2: 21

La gráfica resultante puede verse en la figura 8.31.:

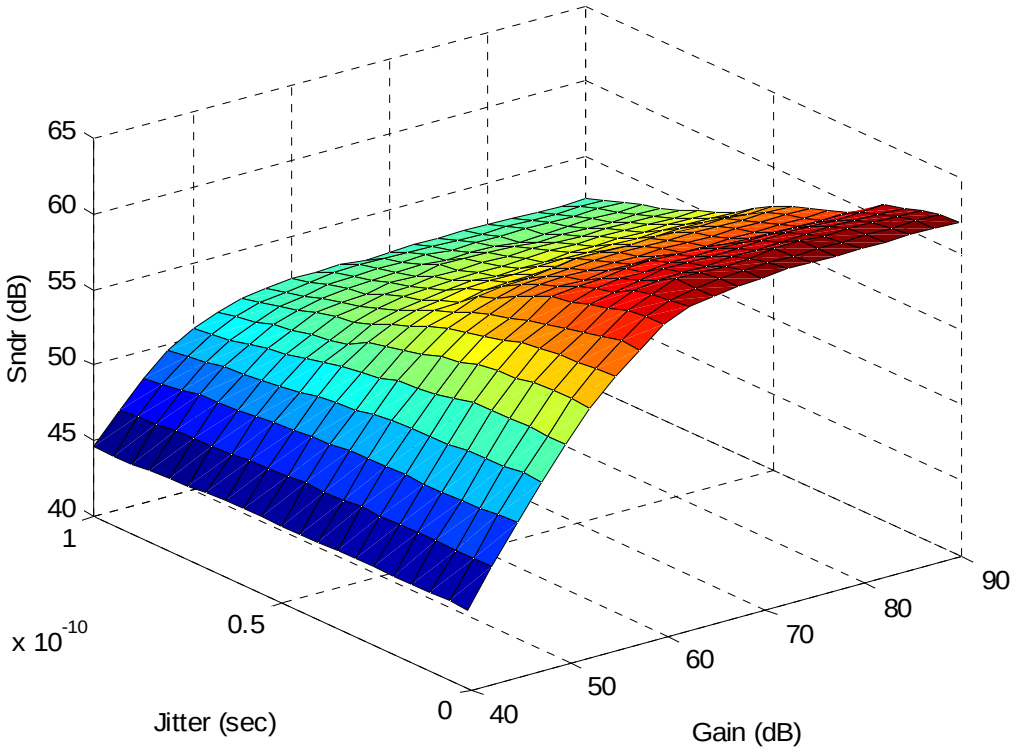


Figura 8.31.: Gráfica de SNDR frente a la Ganancia y al Jitter

- **Pipeline Simulación Nº 20:**

Datos de la simulación:

Tipo: Barrido doble de Ganancia y Amplitud de entrada
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 40
Stop: 90
Points: 21
Start2: 0.1
Stop2: 1
Points2: 21

La gráfica resultante puede verse en la figura 8.32., donde puede apreciarse que la SNDR va subiendo en la dirección en la que aumentan la ganancia y la amplitud de entrada.

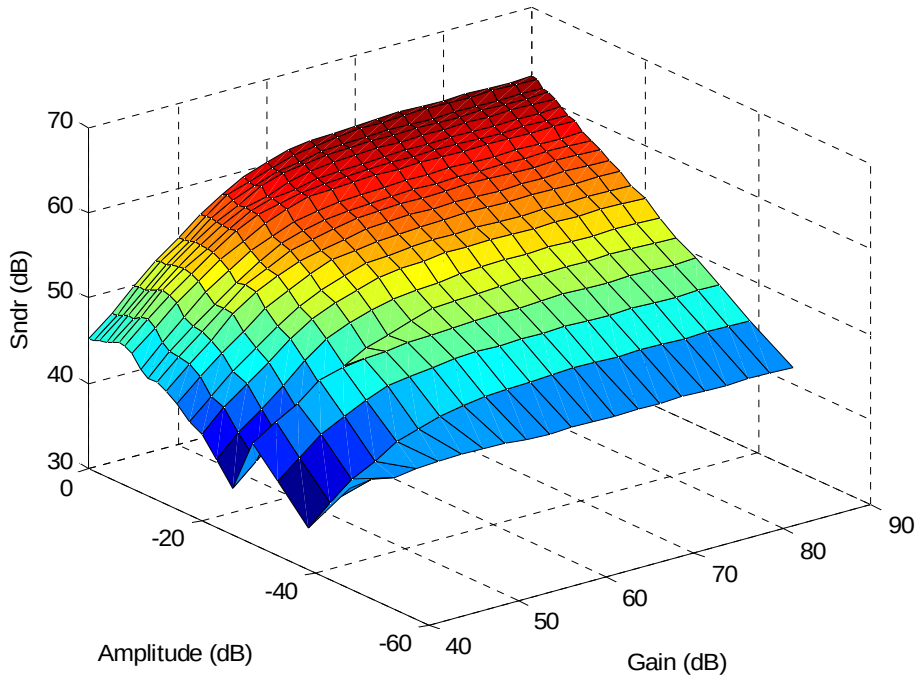


Figura 8.32.: Gráfica de SNDR frente a la Ganancia y a la Amplitud de entrada

- **Pipeline Simulación Nº 21:**

Datos de la simulación:

Tipo: Barrido doble de Ganancia y Full Scale
Entrada: Seno
Modelo: Simulacionnoideal10bitsFiniteGain.mdl
Effort: 7
Start: 40
Stop: 90
Points: 21
Start2: 1
Stop2: 3
Points2: 21

La gráfica resultante puede verse en la figura 8.33., donde puede apreciarse que la SNDR va subiendo en la dirección en la que aumenta la ganancia, cayendo bruscamente cuando FS disminuye de 2V (teniendo una amplitud de entrada de 1V).

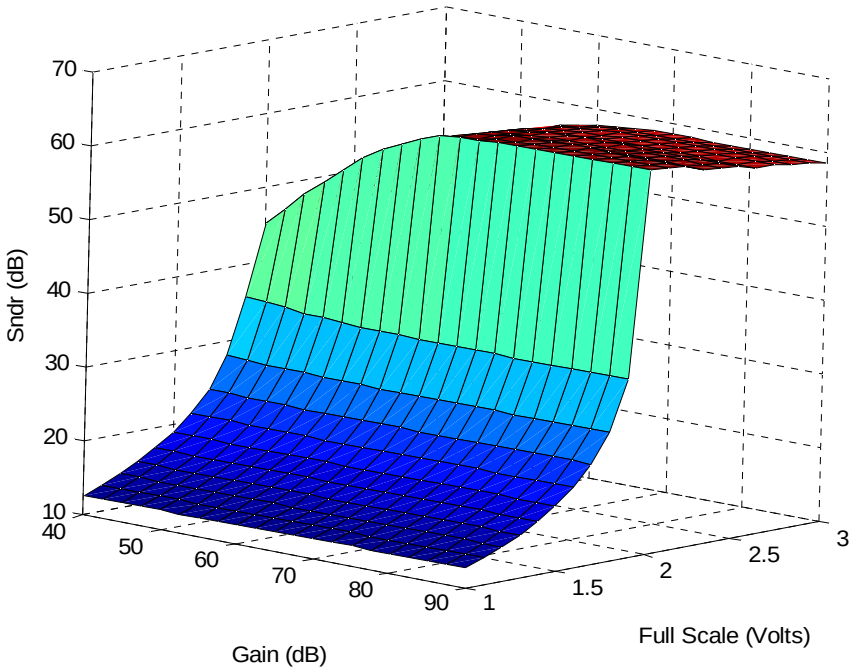


Figura 8.33.: Gráfica de SNDR frente a la Ganancia y FS

8.3 Convertidor flash de 6 bits

Comenzaremos con las simulaciones ideales de este convertidor, después pasaremos a los barridos simples y por último expondremos los barridos dobles:

- **Flash Simulación Nº 1:**

Datos de la simulación:

Tipo: Simulación ideal
Entrada: Rampa
Modelo: Flashwithoffsetandrelay.mdl

El aspecto de la GUI puede verse en la figura 8.34., en la que pueden apreciarse los valores de las variables de entrada (zona de Inputs) y de las no idealidades (zona de Non Idealities), y los resultados siguientes (zona de Results):

MaxINL=0
MaxDNL=0
Missing Codes=0

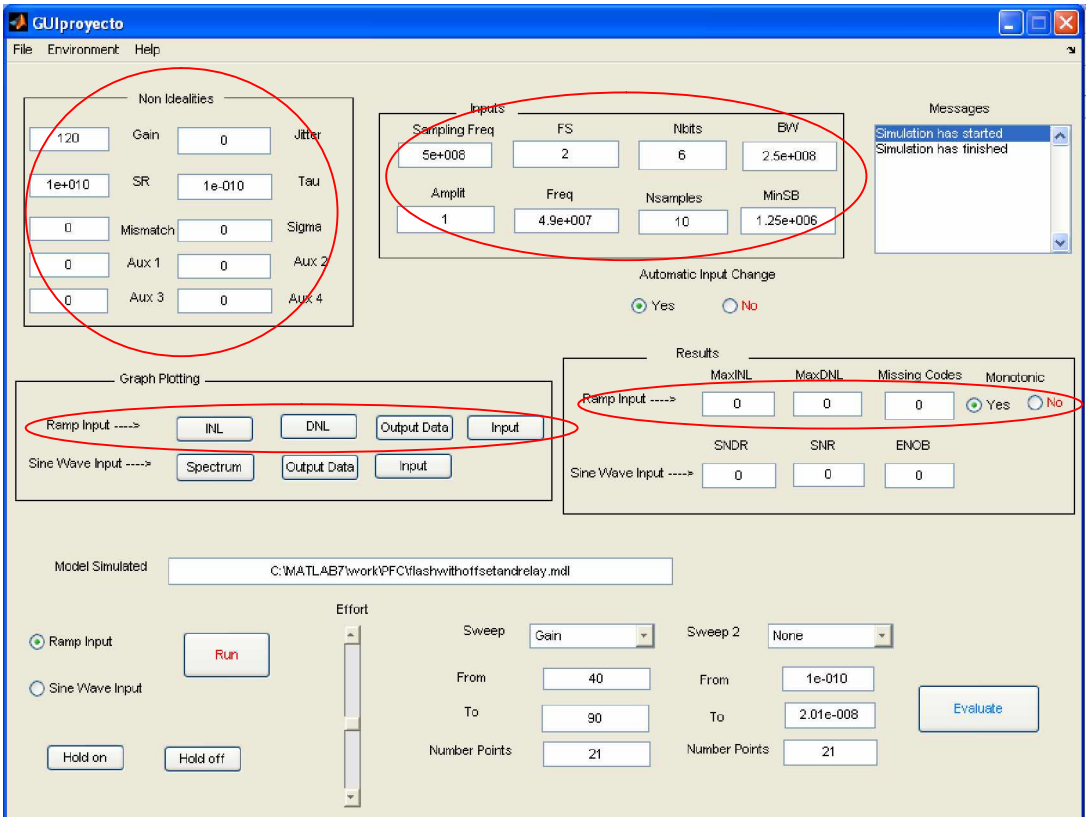


Figura 8.34.: GUI tras Flash Simulación Nº 1

Las gráficas de INL, DNL, Output Data e Input pueden verse en las figuras 8.35., 8.36., 8.37. y 8.38. respectivamente. Puede apreciarse que el valor máximo del valor absoluto de la gráfica de INL coincidirá con el resultado MaxINL (MaxINL tiene en cuenta los códigos perdidos pero, al salir los códigos perdidos 0, coincide el máximo del valor absoluto de la gráfica con MaxINL). De la misma forma, coincide el máximo del valor absoluto de la gráfica de DNL con MaxDNL (en MaxDNL no se tienen en cuenta los códigos perdidos, luego el máximo de la gráfica de DNL siempre coincidirá con MaxDNL).

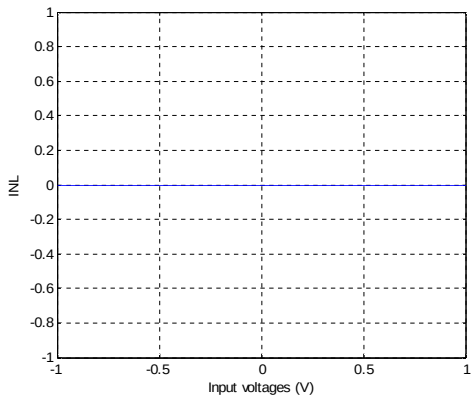


Figura 8.35.: Gráfica INL Flash Simulación Nº 1

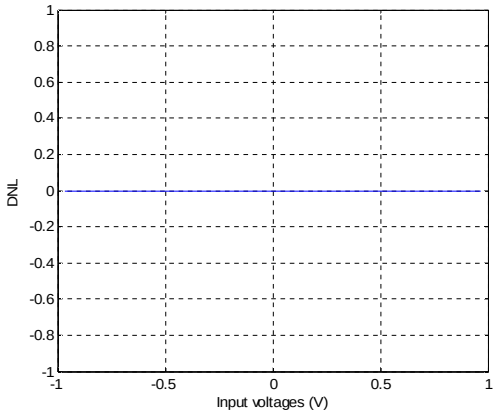


Figura 8.36.: Gráfica DNL Flash Simulación Nº 1

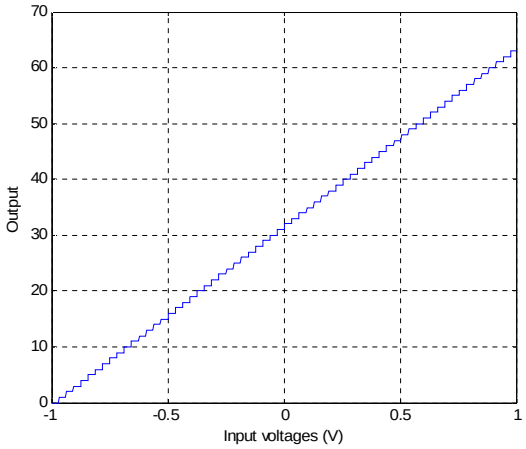


Figura 8.37.: Gráfica Output Data Flash Simulación Nº 1

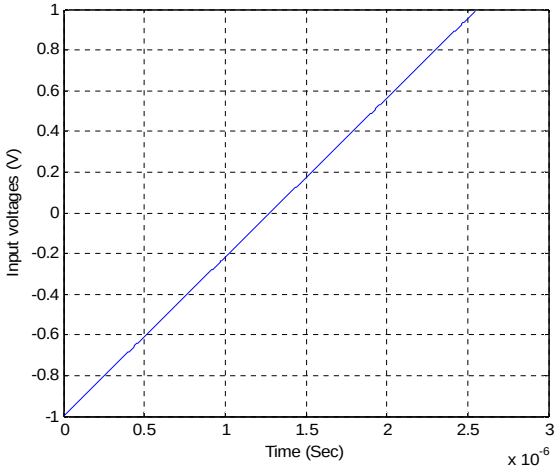


Figura 8.38.: Gráfica Input Flash Simulación Nº 1

- **Flash Simulación Nº 2:**

Datos de la simulación:

Tipo: Simulación ideal
Entrada: Seno
Modelo: Flashwithoffsetandrelay.mdl

El aspecto de la GUI es similar al de la figura 8.34., ya que se han elegido los mismos valores de las variables de entrada y de las no idealidades, sólo hemos cambiado el tipo de entrada, por lo que ahora aparecerán los valores en la zona de Results en los cuadros de texto correspondientes a entrada seno. Se obtienen los resultados siguientes:

SNDR=37.6702
SNR=38.0628
ENOB=5.9651

Las gráficas de Spectrum, Output Data e Input pueden verse en las figuras 8.39., 8.40. y 8.41. respectivamente.

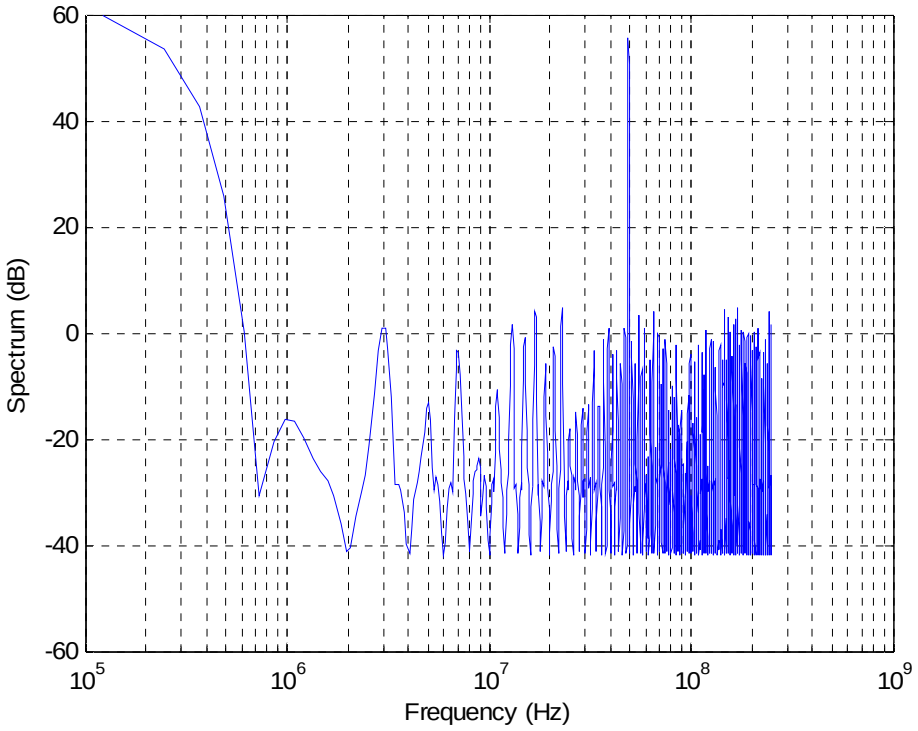


Figura 8.39.: Gráfica Spectrum Flash Simulación Nº 2

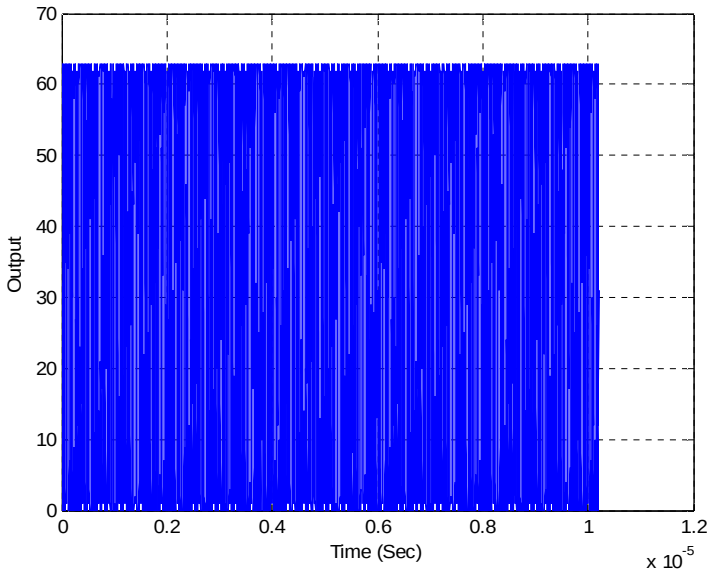


Figura 8.40.: Gráfica Output Data Simulación Nº 2

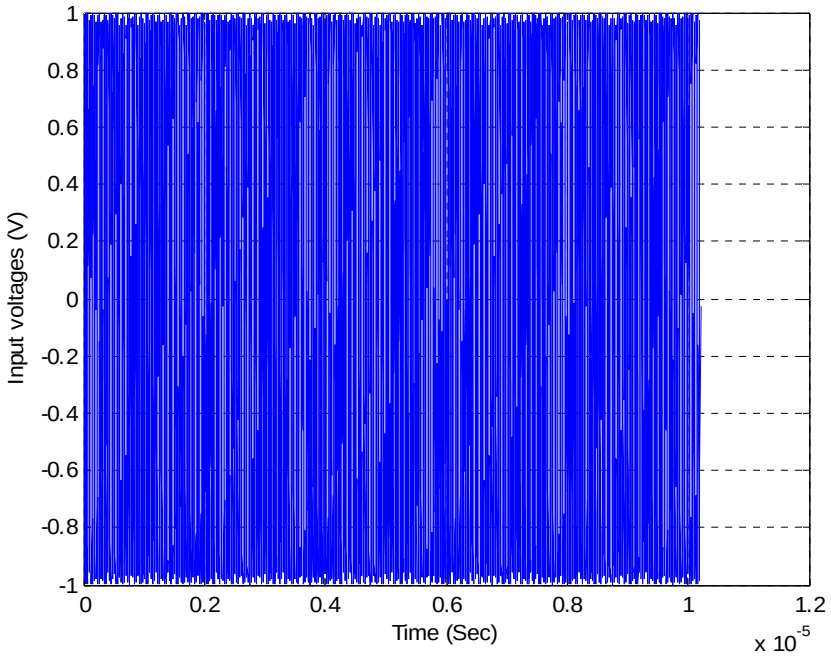


Figura 8.41.: Gráfica Input Flash Simulación Nº 2

- **Flash Simulación Nº 3:**

Datos de la simulación:

Tipo: Barrido simple de offset en las tensiones de referencia (Aux1)
Entrada: Seno
Modelo: Flashwithoffsetandrelay.mdl
Effort: 5
Start: 1e-2
Stop: 1e-3
Points: 11

La gráfica resultante puede verse en la figura 8.42., en la que puede apreciarse cómo, a medida que aumentamos la desviación típica del offset en las tensiones de referencia, la SNDR tiende a disminuir, aunque a veces puede aumentar debido al carácter aleatorio de la variable, pero como hemos dicho en general tiende a disminuir.

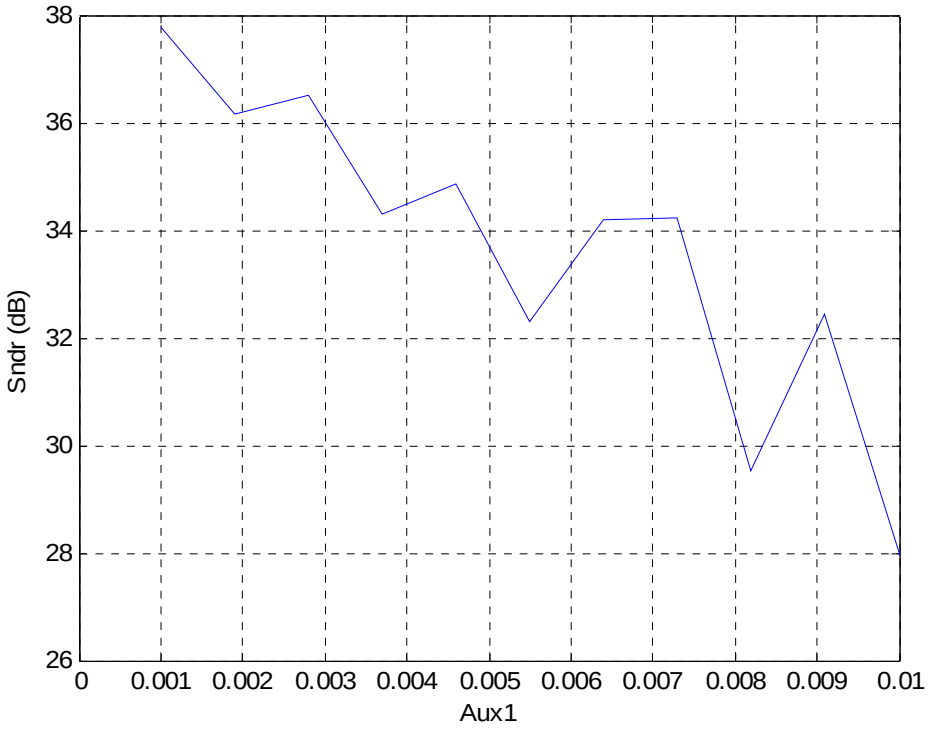


Figura 8.42.: Gráfica SNDR frente a la desviación típica del offset en las tensiones de referencia

- **Flash Simulación Nº 4:**

Datos de la simulación:

Tipo: Barrido simple de offset en los comparadores (Aux2)
Entrada: Seno
Modelo: Flashwithoffsetandrelay.mdl
Effort: 5
Start: 1e-2
Stop: 1e-3
Points: 11

La gráfica resultante puede verse en la figura 8.43., en la que puede apreciarse cómo, a medida que aumentamos la desviación típica del offset en los comparadores, la SNDR tiende a disminuir, aunque a veces puede aumentar debido al carácter aleatorio de la variable, pero como hemos dicho en general tiende a disminuir.

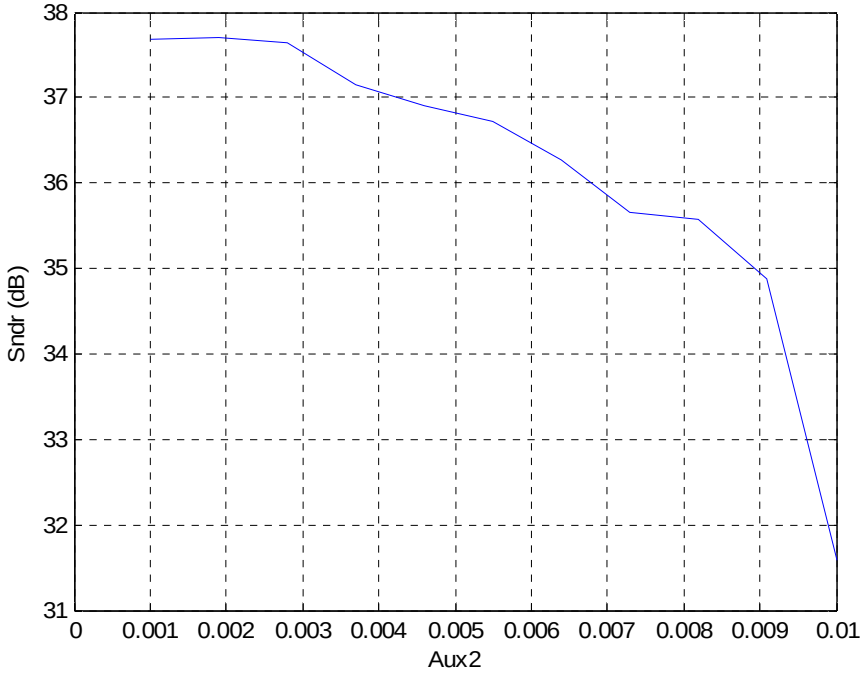


Figura 8.43.: Gráfica SNDR frente a la desviación típica del offset en los comparadores

- **Flash Simulación Nº 5:**

Datos de la simulación:

Tipo: Barrido simple de histéresis en los comparadores (Aux3)
Entrada: Seno
Modelo: Flashwithoffsetandrelay.mdl
Effort: 5
Start: 5e-2
Stop: 1e-3
Points: 11

La gráfica resultante puede verse en la figura 8.44., en la que puede apreciarse cómo, a medida que aumentamos la histéresis en los comparadores, la SNDR tiende a disminuir. El punto de caída de 3dB en la SNDR se produce con un valor de histéresis de alrededor de 0.02.

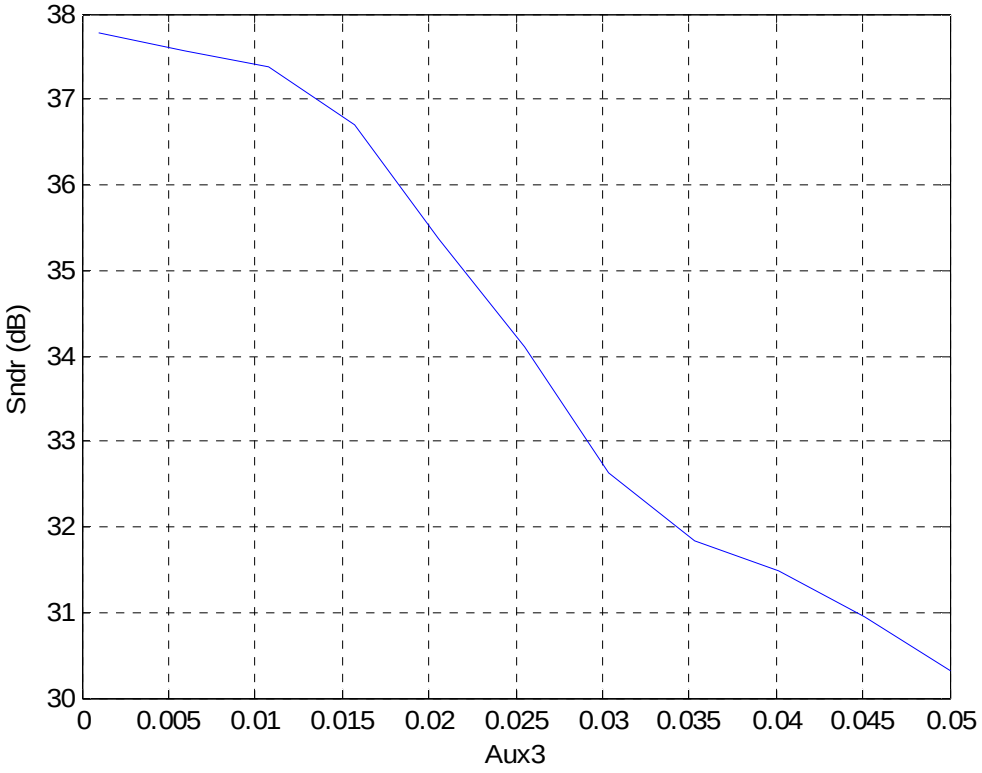


Figura 8.44.: Gráfica SNDR frente a la histéresis en los comparadores

- **Flash Simulación Nº 6:**

Datos de la simulación:

Tipo: Barrido simple de Jitter
Entrada: Seno
Modelo: Flashwithoffsetandrelay.mdl
Effort: 5
Start: 1e-10
Stop: 1e-12
Points: 21

La gráfica resultante puede verse en la figura 8.45., en la que puede apreciarse cómo, a medida que aumentamos el jitter, la SNDR tiende a disminuir. El punto de caída de 3dB en la SNDR se produce con un valor de jjtter de alrededor de 400 picosegundos.

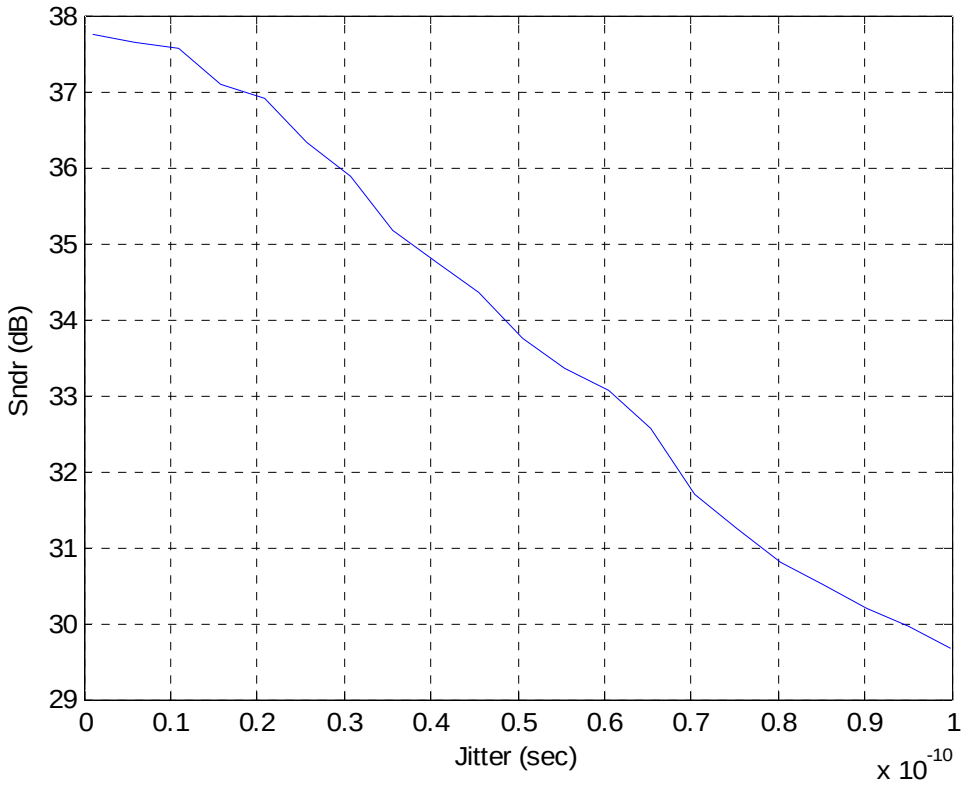


Figura 8.45.: Gráfica SNDR frente al Jitter

- **Flash Simulación Nº 7:**

Datos de la simulación:

Tipo: Barrido doble de offset en las tensiones de referencia (Aux1) e histéresis en los comparadores (Aux3)
Entrada: Seno
Modelo: Flashwithoffsetandrelay.mdl
Effort: 5
Start: 1e-2
Stop: 1e-3
Points: 7
Start2: 5e-2
Stop2: 1e-3
Points2: 7

La gráfica resultante puede verse en la figura 8.46., donde puede apreciarse que la SNDR va subiendo en la dirección en la que disminuyen la histéresis en los comparadores y la desviación típica del offset en las tensiones de referencia, aunque a veces puede disminuir debido al carácter aleatorio del offset, pero como hemos dicho en general tiende a subir en la dirección indicada.

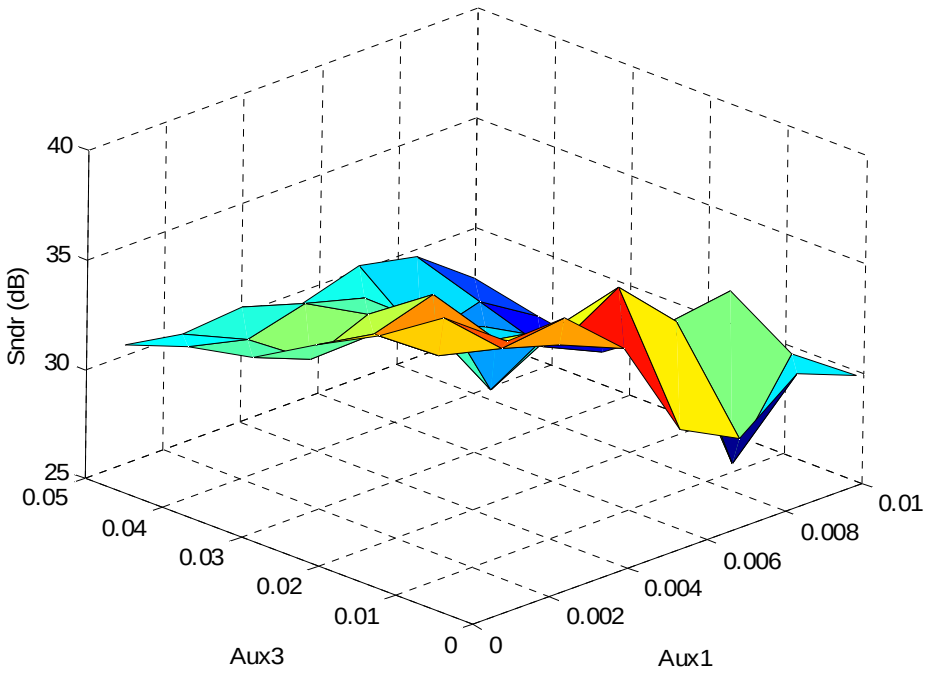


Figura 8.46.: Gráfica SNDR frente a la desviación típica del offset en las tensiones de referencia y a la histéresis en los comparadores

Capítulo 9
Conclusiones y futuras líneas de desarrollo

9.1 Conclusiones

Se ha desarrollado una herramienta de simulación de convertidores A/D. Este proyecto no pretende mostrar una herramienta definitiva sino establecer una base sobre la que poder construir nuevas ampliaciones y mejoras.

Se había fijado como objetivo aportar una herramienta que ayude a simular el comportamiento de los convertidores al cambiar ciertos parámetros como el tiempo de muestreo, el fondo de escala, etc., y también teniendo en cuenta la existencia de no idealidades como la ganancia finita, el desapareamiento entre capacidades, el jitter, el offset, etc., de forma que con este programa se pueda proceder al análisis rápido y sistemático del efecto que producen en el convertidor estudiado todas estas variables. Se ha pretendido satisfacer esta necesidad de simuladores de convertidores mediante una interfaz amigable, versátil, de sencillo e intuitivo manejo y que posibilitara a su vez futuras líneas de desarrollo.

Es por esto que se ha organizado la interfaz en diversos bloques diferenciados, agrupando controles (botones, cuadros de texto, menús desplegables, etc.) relativos a tareas similares, haciendo transparentes al usuario elementos complejos de la simulación (bucles, cálculos matemáticos, etc.).

Se ha hecho especial incidencia en la versatilidad, de modo que el programa va a permitirnos realizar operaciones muy diferenciadas, como análisis simples (consistentes en una única simulación con unos valores determinados de los parámetros), barridos simples (varias simulaciones con distintos valores de un parámetro y el mismo valor del resto), y barridos dobles (como los barridos simples pero con 2 variables). Del mismo modo, podremos seleccionar entrada rampa o seno, ajustar la frecuencia y la amplitud, y guardar y salvar estados (valores de los parámetros).

Un aspecto muy interesante de la herramienta desarrollada es que permite simular cualquier tipo de convertidor con cualquier resolución y con un alto número de

no idealidades que no tienen por qué exactamente las descritas por el programa, sólo necesitaríamos construir el modelo de ese convertidor y llamar a las no idealidades que queramos incluir con los mismos nombres que se les ha dado en el programa (para éste será transparente, es decir, si a un modelo con offset por ejemplo llamamos al offset ganancia podriamos hacer simulaciones asumiendo que los valores que pongamos en ganancia corresponderán al offset). Esto nos permite, en la práctica, simular infinidad de convertidores diferentes.

La herramienta desarrollada está integrada en Matlab, lo que conlleva el aprovechamiento de multitud de herramientas que ofrece Matlab, la posibilidad de acceder a distintas variables a lo largo del proceso y el poder reutilizar las funciones desarrolladas fuera del programa.

Como conclusión final, resumiendo lo anteriormente expuesto, podríamos decir que la versatilidad y la variedad de posibilidades que nos permite esta herramienta, junto a la potencia de Matlab, hace que el programa implementado pueda suponer un elemento muy interesante y de gran ayuda para los diseñadores de convertidores.

9.2 Futuras líneas de desarrollo

A continuación se proponen algunas de las posibles líneas de desarrollo que pueden surgir de este proyecto fin de carrera:

- Introducir nuevas figuras de mérito en la herramienta de simulación, como el rango dinámico, etc.
- Incluir nuevas no idealidades en el programa, como podría ser el efecto de un polo finito en los amplificadores operacionales, etc.
- Implementar todos los barridos dobles posibles (en el programa ya se ha dejado implícitamente implementada la posibilidad de realizar las simulaciones de cualquier barrido doble, sólo sería necesario usar en los modelos las variables auxiliares, cuyos barridos están implementados en todas las combinaciones posibles).
- Ampliar las librerías y los modelos disponibles para el usuario, basándonos en el artículo de Malcovati (referencia sexta de la bibliografía).
- Añadir una función de coste proporcional a la potencia de cada etapa de los convertidores.
- Hacer un estudio completo y exhaustivo de diversos convertidores existentes en la bibliografía, como los semiflash, folding, interleaving, sigma-delta, etc.
- Añadir más mensajes a los que muestra la herramienta, indicando posibles fuentes de error en simulaciones, valores poco lógicos que se le den a las variables,etc.

Bibliografía

- Mikael Gustavsson, J. Jacob Wikner, Nianxiong Nick Tan, **"CMOS Data Converters for Communications"**, Kluwer Academic Publishers, 2000.

- David Johns, Ken Martin, **"Analog Integrated Circuit Design"**, John Wiley and Sons, Inc., 1997.

- Alfi Moscovici, **"High Speed A/D Converters: Understanding Data Converters Through Spice"**, Kluwer Academic Publishers, 2001.

- The MathWorks, Inc., **"Creating Graphical User Interfaces: Versión 7"**, 2004.

- Andrew M. Abo, Paul R. Gray, **"A 1.5-V, 10-bit, 14.3MS/s CMOS Pipeline Analog-to-Digital Converter"**, IEEE Journal of Solid-State Circuits, Vol. 34, Nº 5, Mayo 1999.

- Piero Malcovati, Simona Brigati, Fabrizio Francesconi, Franco Maloberti, Paolo Cusinato, Andrea Baschirotto, **"Behavioral Modeling of Switched-Capacitor Sigma-Delta Modulators"**, IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications, Vol. 50, Nº 3, Marzo 2003.

Apéndice 1
Manual de Usuario

A1.1 Introducción

Este apéndice pretende ser una completa guía para los usuarios de la aplicación. Dicha aplicación es un simulador de convertidores analógicos y digitales, que nos permitirá el análisis sistemático y rápido de dichos convertidores: errores, no idealidades, barridos, etc.

Para instalar esta herramienta de simulación, hemos de copiar en una subcarpeta (creada previamente) de Matlab7\work el contenido de la subcarpeta simconverter de la raíz del CD suministrado (es decir, si por ejemplo tenemos el CD introducido en la unidad D: y Matlab7 instalado en la raíz de C:, habría que copiar todo el contenido de D:\simconverter en C:\Matlab7\work\PFC).

Una vez hayamos instalado la herramienta, es decir, hayamos copiado el contenido del CD, y tras abrir Matlab, situarnos en el directorio en el que hayamos copiado la aplicación, y colocar el cursor en la ventana de comandos, tras hacer todo esto ya podemos usar la orden *simconverter* para arrancar el simulador. El resultado es la aparición de la ventana que se muestra en la figura A1.1., que describiremos con detalle en los siguientes apartados.

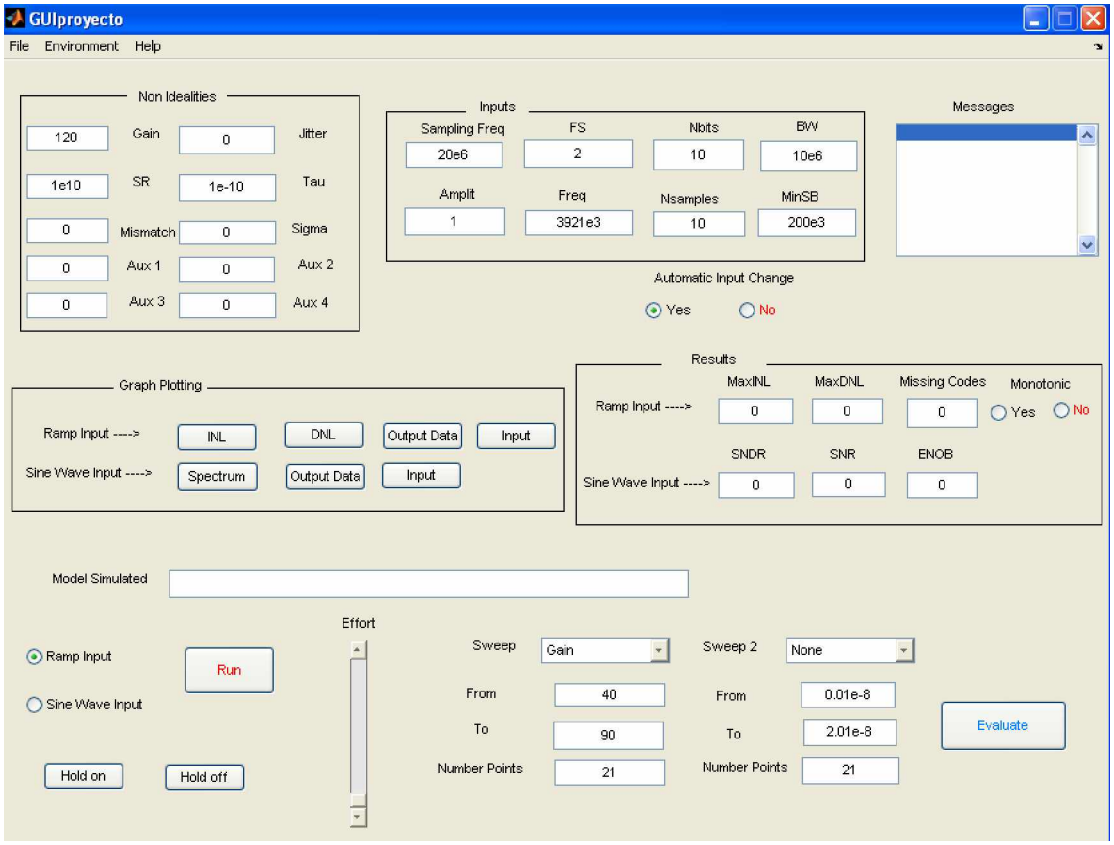


Figura A1.1.: GUI del simulador

A1.2 Estructura de la GUI

En este apartado, vamos a explicar con detalle la estructura de la GUI de la aplicación implementada.

Como puede apreciarse en la figura A1.1., a grandes rasgos podemos distinguir 7 zonas o partes principales en la GUI, que mencionaremos aquí y describiremos con más detalle en los siguientes sub-apartados:

- Zona de Menús: Situada en la parte superior de la ventana, contiene 3 menús: File, Environment y Help.
- Zona de Non Idealities: Debajo de los menús a la izquierda, contiene los valores de las no idealidades para los estudios simples.
- Zona de Inputs: Debajo de los menús en el centro, contiene los valores de las entradas, son generales para todas las simulaciones. También presenta la opción de cambio de valores automáticos.
- Zona de Messages: Debajo de los menús a la derecha, contiene los mensajes de progreso en las simulaciones que genera la aplicación.

- Zona de Graph Plotting: Debajo de la Zona de Non Idealities, contiene las posibles gráficas que podemos pedirle al programa que dibuje.

- Zona de Results: Debajo de la Zona de Inputs, contiene los valores de salida que genera el programa.

- Zona de Simulación: Situada en la parte inferior de la ventana, contiene todas las opciones de simulación, tanto los estudios simples como los barridos.

A1.2.1. Zona de Menús

La Zona de Menús, como puede verse en la figura A1.1., está compuesta por 3 menús diferentes: File, Environment y Help. Veremos con detalle cada uno de estos menús en los siguientes sub-apartados.

A1.2.1.1 Menú File

El menú File, como su nombre indica, permitirá el tratamiento de archivos por parte del programa. Podemos apreciar las opciones que nos proporciona el menú File en la figura A1.2.:

- New: Crea un archivo nuevo.

- Open: Abre un archivo existente.

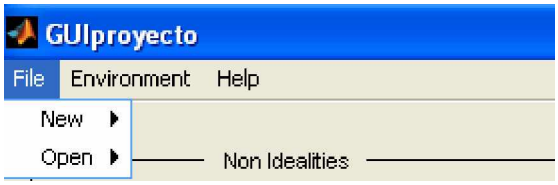


Figura A1.2.: Menú File

Vemos estas opciones con más detalle a continuación.

A1.2.1.1.1 Menú File Submenú New:

El contenido del submenú New puede verse en la figura A1.3.

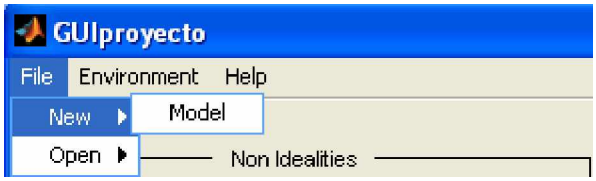


Figura A1.3.: Menú File Submenú New

La única opción de este submenú es Model. Al seleccionarla, se producen 2 efectos:

- Se abre el archivo current.mdl del directorio de la aplicación. Dicho archivo es el molde que usaremos para crear nuestros modelos de simulación, porque sólo contiene un generador de señal y un Scope para visualizar, quedando el resto vacío para que el usuario introduzca ahí el modelo que desea simular. El aspecto de este archivo puede verse en la figura A1.4.

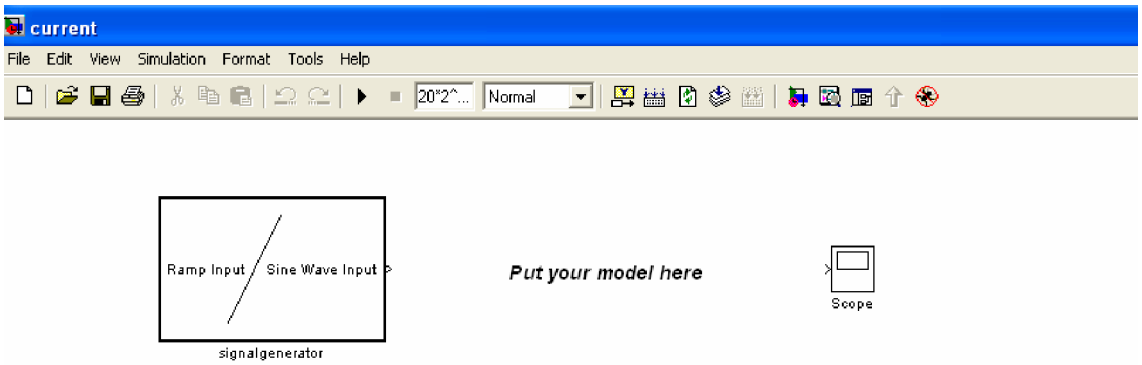


Figura A1.4.: Archivo current.mdl

- Aparece en el cuadro de texto Model Simulated de la Zona de Simulación la ruta completa del modelo, como puede apreciarse en la figura A1.5.

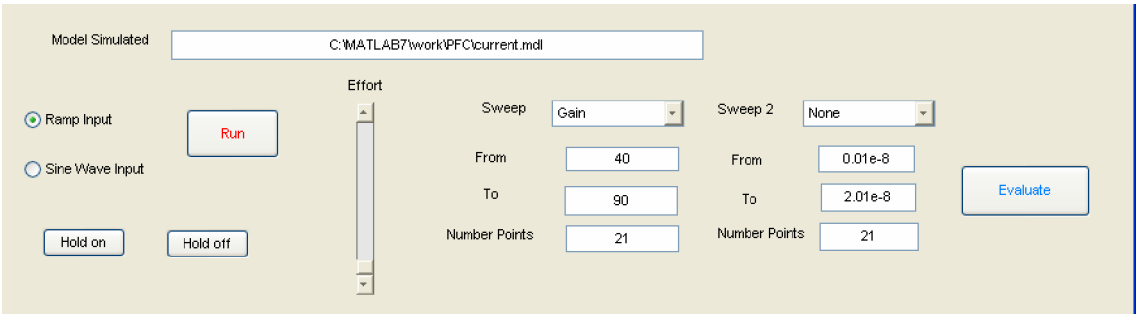


Figura A1.5.: Zona de Simulación tras elegir File->New->Model

A1.2.1.1.2 Menú File Submenú Open:

El contenido del submenú Open puede verse en la figura A1.6.

Figura A1.6.: Menú File Submenú Open

Este submenú presenta 2 opciones distintas:

- Library: Al elegir esta opción, nos aparece el cuadro de diálogo de la figura A1.7., en el que habrá que seleccionar la librería que queremos abrir. En este cuadro de diálogo, se muestran todos los archivos .mdl del subdirectorio Library del directorio de trabajo.

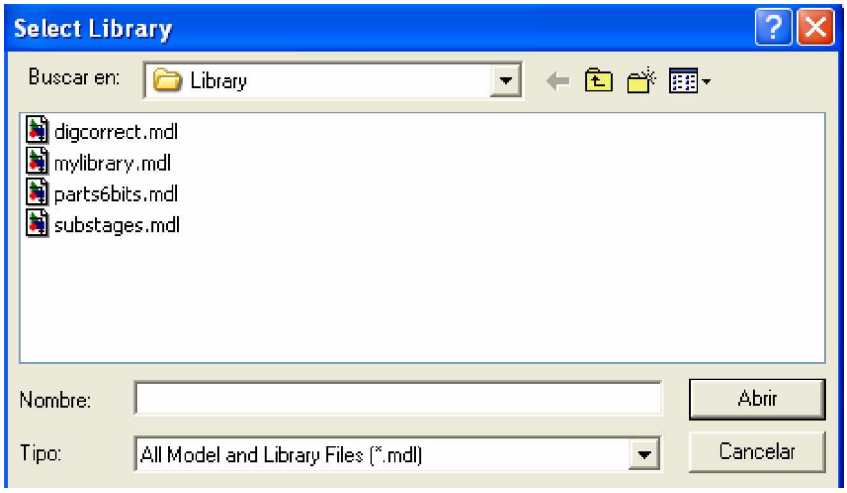


Figura A1.7.: Cuadro de diálogo al elegir File->Open->Library

- Model: Al elegir esta opción, nos aparece el cuadro de diálogo de la figura A1.8., en el que habrá que seleccionar el modelo que queremos abrir para simularlo. En este cuadro de diálogo, se muestran todos los archivos .mdl del directorio de trabajo. Cuando se elige uno de estos modelos, se abre y además su ruta completa aparece en el cuadro de texto Model Simulated de la Zona de Simulación.

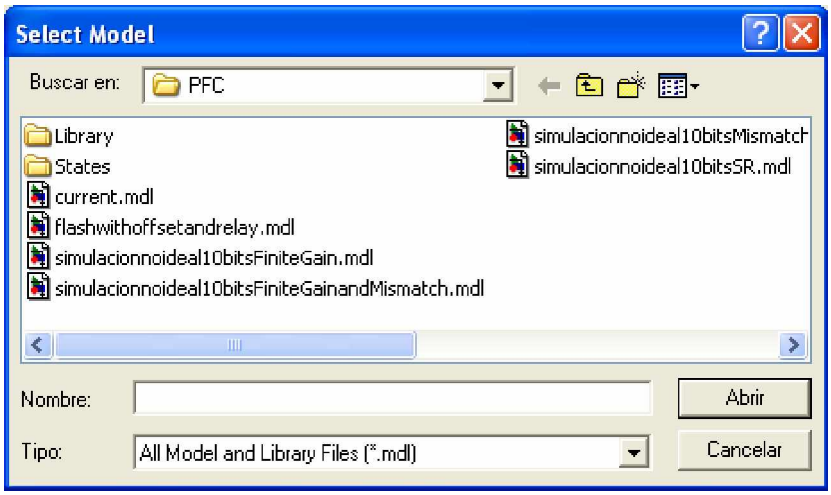


Figura A1.8.: Cuadro de diálogo al elegir File->Open->Model

A1.2.1.2 Menú Environment

El menú Environment permitirá al programa cargar y guardar estados (valores de las variables, barridos, etc.) del mismo, como veremos a continuación. Estos estados se guardarán en archivos .m, porque la carga del estado consistirá en la ejecución del script (archivo .m) que lo almacena. Podemos apreciar las opciones que nos proporciona este menú en la figura A1.9.:

Figura A1.9.: Menú Environment

Vemos estas opciones con más detalle a continuación:

- Load: Carga un estado existente. Al elegir esta opción, nos aparece el cuadro de diálogo de la figura A1.10., en el que habrá que seleccionar el estado que queremos cargar. En este cuadro de diálogo, se muestran todos los archivos .m del subdirectorio States del directorio de trabajo.

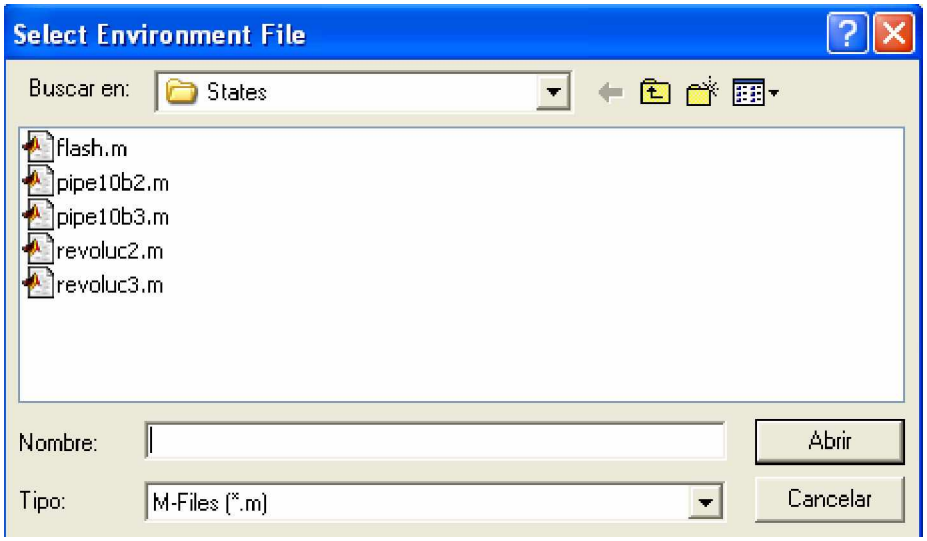


Figura A1.10.: Cuadro de diálogo al elegir Environment->Load

- Save: Guarda el estado actual. Al elegir esta opción, nos aparece el cuadro de diálogo de la figura A1.11., en el que habrá que seleccionar el nombre del archivo (podemos elegir uno ya existente o especificar un nombre nuevo) en el que queremos almacenar el estado actual. En este cuadro de diálogo, se muestran todos los archivos .m del subdirectorio States del directorio de trabajo.

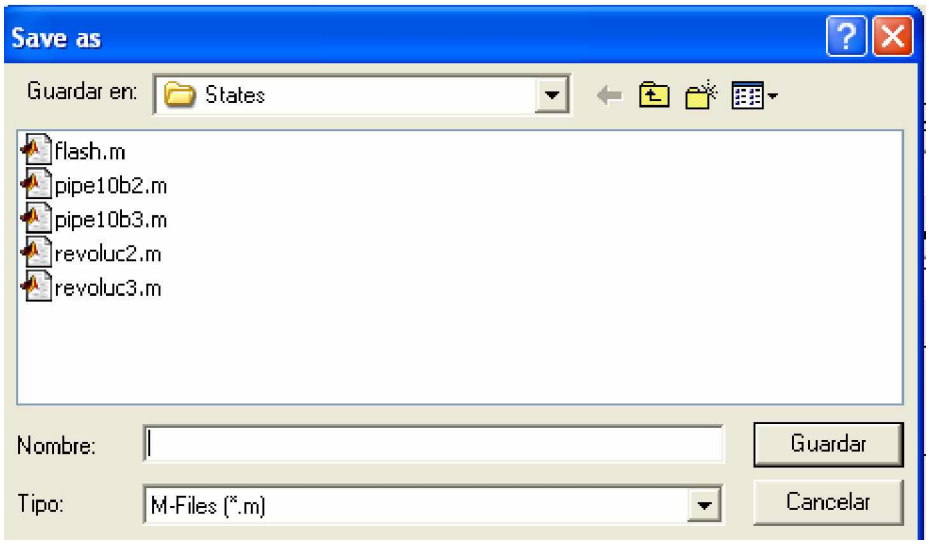


Figura A1.11.: Cuadro de diálogo al elegir Environment->Save

A1.2.1.3 Menú Help

El menú Help nos permitirá hacer 2 cosas distintas, que pueden apreciarse en la figura A1.12.: por un lado, la opción "User Guide" nos abre un pdf con el contenido de este manual para que el usuario reciba una ayuda on-line de forma rápida y sencilla; y por otro lado, la opción "About" nos presenta información de la versión del simulador ante la que se encuentra el usuario.

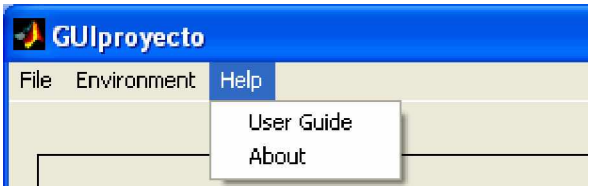


Figura A1.12.: Menú Help

A1.2.2. Zona de Non Idealities

La Zona de Non Idealities está compuesta por una serie de cuadros de texto en los que aparecen los valores de las no idealidades que pueden ser consideradas en el modelo a simular, y cuyo valor puede ser modificado editando dichos cuadros de texto. La estructura de esta zona puede verse en la figura A1.13.

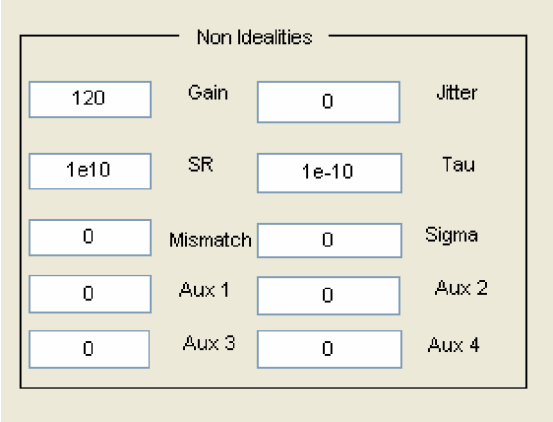


Figura A1.13.: Zona de Non Idealities

Veamos a qué variable (no idealidad) representa cada cuadro de texto:

- Gain: Representa la ganancia finita del amplificador operacional en dB. La variable correspondiente es AodB.

- Jitter: Representa el jitter en el tiempo de muestreo. La variable correspondiente es sigmajitter.
- SR: Representa el slew rate del amplificador operacional. La variable correspondiente es SR.

- Tau: Representa la constante de tiempo en segundos. La variable correspondiente es Tau.

- Mismatch: Representa el desapareamiento entre las capacidades en %. La variable correspondiente es Eps.

- Sigma: Representa la desviación típica del offset en los DACs. La variable correspondiente es sigma, de la cual se calculan (como números aleatorios de media 0 y desviación típica sigma) dos variables más: el offset para la tensión de referencia positiva y negativa (VoffsetVrefpn), y el offset para la tensión de referencia nula (Voffset0).

- Aux 1, Aux 2, Aux 3, Aux 4: Variables genéricas que representan otras no idealidades distintas de las ya reseñadas que podríamos incluir en el modelo a simular. Así, en el modelo del convertidor flash de 6 bits que se ha implementado en el programa, usamos Aux1 para el offset de las tensiones de referencia, Aux2 para el offset de los comparadores, y Aux3 para la histéresis de los comparadores.

A1.2.3. Zona de Inputs

La Zona de Inputs está compuesta por una serie de cuadros de texto, en los que aparecen los valores de las variables de entrada y cuyo valor puede ser modificado editando dichos cuadros de texto, y dos radiobuttons que permiten elegir si se desea que al cambiar ciertos valores puedan modificarse otros de forma automática. La estructura de esta zona puede verse en la figura A1.14.

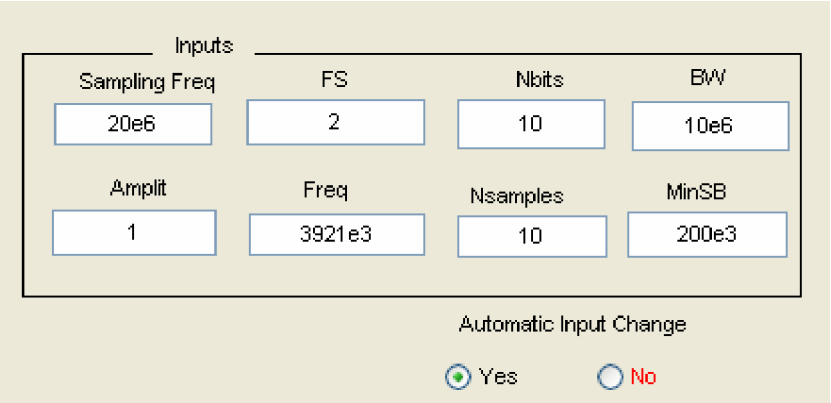


Figura A1.14.: Zona de Inputs

Veamos a qué variable de entrada representa cada cuadro de texto:

- Sampling Freq: Representa la frecuencia de muestreo en hercios. La variable correspondiente será Ts, que representará al tiempo de muestreo (por tanto, será la inversa de la frecuencia de muestreo). Al modificarlo, si está activada la opción de cambio automático de variables, se modifican los cuadros de texto BW, Freq y MinSB (y sus variables correspondientes), con las siguientes ecuaciones:

$$BW = \frac{1}{2Ts}$$
$$Freq = \frac{1}{5.1Ts}$$
$$MinSB = \frac{1}{400Ts}$$

Por último, señalar que, al modificar el tiempo de muestreo, estaremos cambiando también el tiempo de simulación en caso de entrada rampa, ya que éste es de $4 \cdot Effort \cdot 2^N \cdot Ts \cdot \left(1 - \frac{2^{-N}}{4}\right)$, siendo N el número de bits esperados, y Effort una variable que representa el "esfuerzo" en la simulación (influye en el tiempo de la simulación, y por tanto en la rapidez de realización de los cálculos por parte del programa).

- FS: Representa el fondo de escala en voltios de los ADCs y DACs de cada etapa. También va a influir en el rango de la señal de entrada rampa, que va de (-FS/2) a FS/2. La variable correspondiente será FS. Al modificar FS, si está activo el cambio automático, se modifica el cuadro de texto Amplit (y la variable correspondiente) con la siguiente ecuación:

$$Amplit = \frac{FS}{2}$$

- Nbits: Representa el número de bits esperados del convertidor. La variable correspondiente será N. Al modificar esta variable, cambiaremos el tiempo de simulación ($4 \cdot Effort \cdot 2^N \cdot Ts \cdot \left(1 - \frac{2^{-N}}{4}\right)$) en caso de entrada rampa.

- BW: Representa el ancho de banda en hercios que tomaremos para la señal en caso de entrada seno. La variable correspondiente será bandwidth.

- Amplit: Representa la amplitud en voltios de la señal de entrada seno. La variable correspondiente será A.

- Freq: Representa la frecuencia en hercios de la señal de entrada seno. Al modificar la frecuencia de la señal de entrada seno, estaremos cambiando también el tiempo de simulación en caso de entrada seno, ya que éste es de $\frac{10 \cdot (Effort^2 + 1)}{Freq}$.

- Nsamples: Representa el número de muestras alrededor de la frecuencia de la señal que tomaremos para calcular la potencia de dicha señal al hacer cálculos de SNR y SNDR. La variable correspondiente será nsamples.

- MinSB: Representa la banda mínima de señal en hercios, es decir, el valor inferior de frecuencia a partir del cual calcularemos el ruido (hay que coger un valor inferior para que no aparezca ruido flicker). La variable correspondiente será minsignalband.

- Automatic Input Change Yes/No: Son 2 radiobuttons que indican, al estar uno de ellos encendido, si se efectuarán modificaciones automáticas en el valor de ciertas variables de entrada al cambiar el valor de Sampling Freq y de FS. Aparece por defecto seleccionado el valor Yes. La variable correspondiente será Auto.

A1.2.4. Zona de Messages

La Zona de Messages está compuesta por una list box en la cual aparecen los mensajes que genera la aplicación (simulación iniciada, simulación concluida, barrido iniciado, barrido en proceso, barrido finalizado). La estructura de esta zona puede verse en la figura A1.15. (esta zona aparece vacía al arrancar el programa, al ejecutar simulaciones simples o barridos nos irán apareciendo los mensajes mencionados en cada caso).

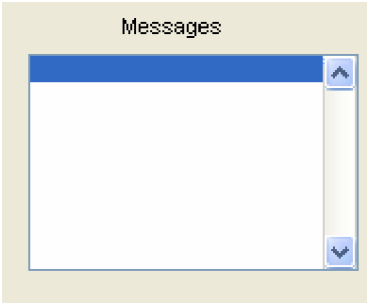


Figura A1.15.: Zona de Messages

A1.2.5. Zona de Graph Plotting

La Zona de Graph Plotting está compuesta por una serie de botones que nos servirán para controlar la gráficas que se quiere visualizar. La estructura de esta zona puede verse en la figura A1.16.

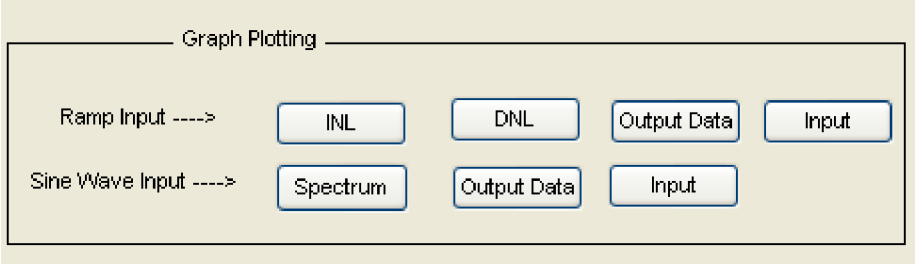


Figura A1.16.: Zona de Graph Plotting

La orden que realiza cada uno de los botones es la siguiente:

- INL: Este botón sólo tiene sentido pulsarlo tras haber hecho una simulación simple con entrada rampa. En ese caso, mostrará una gráfica de INL (error de no linealidad integral, del inglés "Integral Nonlinearity") frente a la señal de entrada en voltios. Cabe destacar que la gráfica de INL que se muestra no tiene en cuenta los códigos perdidos (sí se tiene en cuenta en el cuadro MaxINL de la Zona de Results).

- DNL: Este botón sólo tiene sentido pulsarlo tras haber hecho una simulación simple con entrada rampa. En ese caso, mostrará una gráfica de DNL (error de no linealidad diferencial, del inglés "Differential Nonlinearity") frente a la señal de entrada en voltios.

- Output Data: Aparecen 2 botones denominados Output Data, tendremos que usar el correspondiente a la entrada elegida. En el caso de entrada rampa, mostrará una gráfica de la señal de salida del convertidor (magnitud adimensional que va de 0 a $2^N - 1$) frente a la señal de entrada en voltios. En el caso de entrada seno, mostrará una gráfica de la señal de salida frente al tiempo en segundos.

- Input: Aparecen 2 botones denominados Input, tendremos que usar el correspondiente a la entrada elegida. En ambos casos, se representa la señal de entrada en voltios frente al tiempo en segundos.

- Spectrum: Este botón sólo tiene sentido pulsarlo tras haber hecho una simulación simple con entrada seno. En ese caso, mostrará una gráfica de la PSD en dB frente a la frecuencia en Hz en escala logarítmica.

A1.2.6. Zona de Results

La Zona de Results tendrá utilidad en las simulaciones simples. Está compuesta por una serie de cuadros de texto y 2 radiobuttons, en los que aparecen diversos parámetros de interés para analizar cómo de bueno es el convertidor que estamos simulando y cómo le afectan las no idealidades que estemos considerando. La estructura de esta zona puede verse en la figura A1.17.

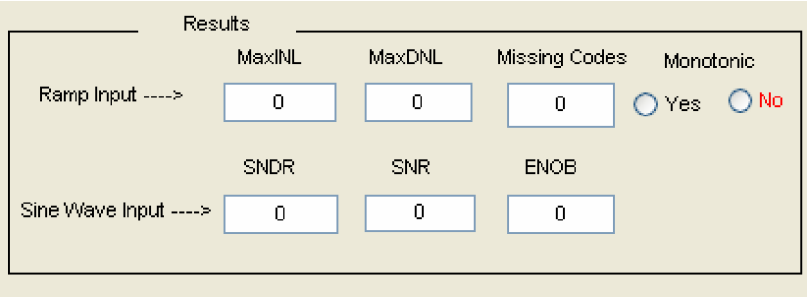


Figura A1.17.: Zona de Results

Vemos a continuación los parámetros a los que hacíamos referencia anteriormente:

- MaxINL: Sólo tiene sentido en una simulación simple con entrada rampa. En este caso, representa la INL máxima del convertidor (incluidos los códigos perdidos).

- MaxDNL: Sólo tiene sentido con entrada rampa, y representa la DNL máxima del convertidor.

- Missing Codes: Sólo tiene sentido con entrada rampa, y representa los códigos perdidos del convertidor.

- Monotonic Yes/No: Sólo tienen sentido con entrada rampa. Son 2 radiobuttons que indican, al estar uno de ellos encendido, si la salida es monotónica o no (aparecen por defecto apagados al arrancar el programa, pero al realizar una simulación simple con entrada rampa ya se enciende uno de los 2).

- SNDR: Sólo tiene sentido en una simulación simple con entrada seno. En este caso, representa la relación señal a ruido y distorsión de la señal de salida en dB.

- SNR: Sólo tiene sentido con entrada seno, y representa la relación señal a ruido de la señal de salida en dB.

- ENOB: Sólo tiene sentido con entrada seno, y representa el número efectivo de bits del convertidor.

A1.2.7. Zona de Simulación

La Zona de Simulación es el corazón del programa, porque desde aquí van a darse las órdenes generales de simulación para realizar los distintos análisis. La estructura de esta zona puede verse en la figura A1.18. A grandes rasgos (describiremos cada elemento con más detalle a continuación), está compuesta por cuadros de texto (modelo a simular, y parámetros de los barridos), radiobuttons (selección de la entrada), pushbuttons (hold, simulación simple y barridos), pop-up menus (selección de barridos), y una slider o barra deslizante (selección de "esfuerzo" en los cálculos de la simulación).

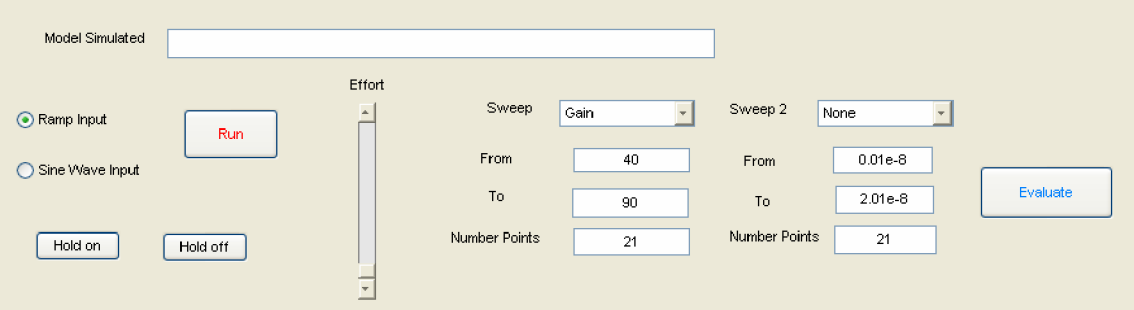


Figura A1.18.: Zona de Simulación

Los elementos de la zona de simulación son:

- Model Simulated: Cuadro de texto en el que aparece el nombre del modelo que se está simulando actualmente (al arrancar el programa, aparece en blanco, lo cual quiere decir que el usuario lo primero que deberá hacer será cargar un modelo usando el menú File->Open->Model). La variable en la que se almacenará el nombre del modelo simulado será modelsimulated.

- Ramp Input / Sine Wave Input: Son 2 radiobuttons que permiten la selección de la señal de entrada (al arrancar el programa, aparece seleccionada entrada rampa). La variable correspondiente será Select.

- Run: Botón que ordena la realización de un análisis simple del modelo indicado en Model Simulated, con el tipo de entrada seleccionado por los radiobuttons Ramp Input y Sine Wave Input.

- Hold on: Botón que, al pulsarse, hace que se dibujen todas las gráficas en los mismos ejes hasta que se pulse Hold off (al arrancar el programa, cada gráfica se dibujará en ejes diferentes).

- Hold off: Botón que, al pulsarse, hace que se dibuje cada gráfica en un eje diferente hasta que se pulse Hold on.

- Effort: Barra deslizante que controla el "esfuerzo" en los cálculos, esto es, el tiempo de simulación, lo cual nos va a repecutir en la rapidez con

la que efectuaremos nuestras simulaciones (a mayor tiempo de simulación, más lentas serán nuestras simulaciones).

- Sweep: Menú desplegable que se utiliza para realizar barridos. Si en el menú desplegable Sweep 2 se elige None, se efectuará un barrido simple, de lo contrario se efectúa un barrido doble (hemos de tener en cuenta que no debemos elegir en Sweep 2 el mismo valor que en Sweep, porque no tiene sentido realizar un barrido doble de la misma variable). Las opciones de Sweep se muestran en la figura A1.19 (al arrancar el programa, aparece por defecto seleccionado Gain). Cabe destacar también que, según el barrido elegido, cambian automáticamente los parámetros del barrido a unos valores estándar para cada caso. La variable correspondiente será val.

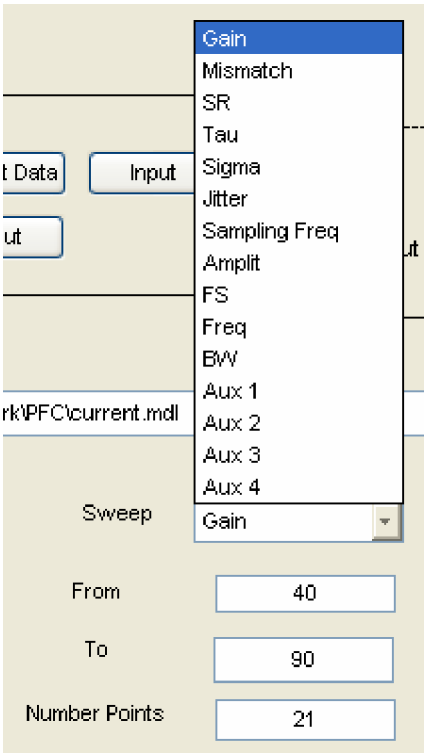


Figura A1.19.: Opciones de Sweep

- Sweep 2: Menú desplegable que nos ofrece la posibilidad de realizar barridos dobles (en caso de elegir None se realizaría un barrido simple). Las opciones de Sweep 2 se muestran en la figura A1.20. (al arrancar el programa, aparece por defecto seleccionado None). Cabe destacar también que, según el barrido elegido, cambian automáticamente los parámetros del barrido a unos valores estándar para cada caso. La variable correspondiente será val2.

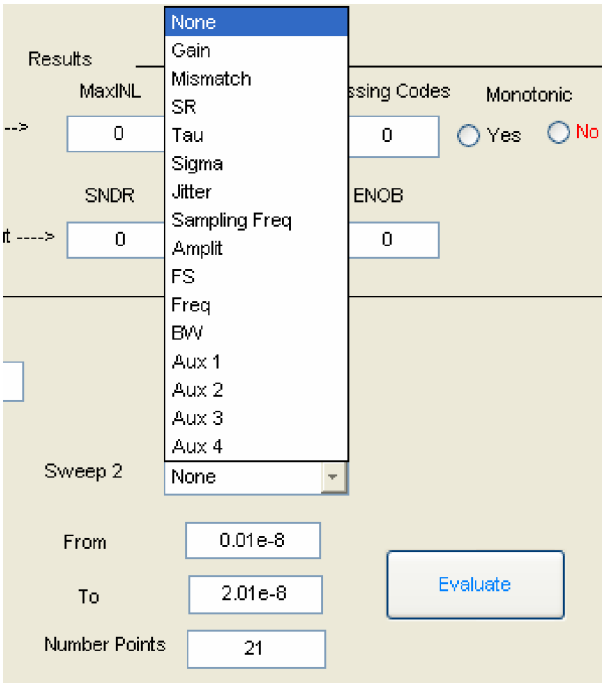


Figura A1.20.: Opciones de Sweep 2

- From: Cuadro de texto que aparece con Sweep (variable Start) y Sweep 2 (variable Start2). En cada caso, indica el valor inicial de la variable para el barrido.

- To: Cuadro de texto que aparece con Sweep (variable Stop) y Sweep 2 (variable Stop2) . En cada caso, indica el valor final de la variable para el barrido.

- Number Points: Cuadro de texto que aparece con Sweep (variable Points) y Sweep 2 (variable Points2). En cada caso, indica el número de puntos (valores distintos de la variable) para el barrido.

- Evaluate: Botón que ordena la realización de un barrido (del tipo seleccionado por Sweep y Sweep 2) del modelo indicado en Model Simulated, con el tipo de entrada seleccionado por los radiobuttons Ramp Input y Sine Wave Input. Al concluir el barrido, dependiendo de la entrada y de si es barrido simple o doble, aparece una determinada gráfica: con entrada rampa, si es barrido simple la gráfica es de INL máxima (incluyendo los códigos perdidos) frente a la variable que varía, pero si es barrido doble la gráfica será de INL máxima frente a las 2 variables que varían (en 3 dimensiones); con entrada seno, las gráficas son análogas pero con SNDR en dB. Se han implementado todos los barridos simples, y la relación de barridos dobles implementados puede verse en las tablas de las figuras A1.21. y A1.22. (en las filas se han representado los valores de Sweep, en las columnas los valores de Sweep 2):

	Gain	Mis match	SR	Tau	Sigma	Jitter	Ts	Amplit
Gain	NO	***SI***	NO	NO	***SI***	***SI***	***SI***	***SI***
Mis match	NO	NO	NO	NO	***SI***	***SI***	***SI***	***SI***
SR	NO	NO	NO	***SI***	***SI***	***SI***	NO	NO
Tau	NO	NO	NO	NO	***SI***	***SI***	NO	NO
Sigma	NO	NO	NO	NO	NO	***SI***	NO	NO
Jitter	NO	NO	NO	NO	NO	NO	NO	NO
Sam pling Freq	NO	NO	NO	NO	NO	NO	NO	NO
Amplit	NO	NO	NO	NO	NO	NO	NO	NO
FS	NO	NO	NO	NO	NO	NO	NO	NO
Freq	NO	NO	NO	NO	NO	NO	NO	NO
BW	NO	NO	NO	NO	NO	NO	NO	NO
Aux 1	NO	NO	NO	NO	NO	NO	NO	NO
Aux 2	NO	NO	NO	NO	NO	NO	NO	NO
Aux 3	NO	NO	NO	NO	NO	NO	NO	NO
Aux 4	NO	NO	NO	NO	NO	NO	NO	NO

Figura A1.21.: Primera parte de la tabla de barridos dobles

	Sam pling Freq	Freq	BW	Aux 1	Aux 2	Aux 3	Aux 4
Gain	***SI***	***SI***	NO	NO	NO	NO	NO
Mis match	***SI***	***SI***	NO	NO	NO	NO	NO
SR	NO	NO	NO	NO	NO	NO	NO
Tau	NO	NO	NO	NO	NO	NO	NO
Sigma	NO	NO	NO	NO	NO	NO	NO
Jitter	NO	NO	NO	NO	NO	NO	NO
Sam pling Freq	NO	NO	NO	NO	NO	NO	NO
Amplit	NO	NO	NO	NO	NO	NO	NO
FS	NO	NO	NO	NO	NO	NO	NO
Freq	NO	NO	NO	NO	NO	NO	NO
BW	NO	NO	NO	NO	NO	NO	NO
Aux 1	NO	NO	NO	NO	***SI***	***SI***	***SI***
Aux 2	NO	NO	NO	NO	NO	***SI***	***SI***
Aux 3	NO	NO	NO	NO	NO	NO	***SI***
Aux 4	NO	NO	NO	NO	NO	NO	NO

Figura A1.22.: Segunda parte de la tabla de barridos dobles

A1.3 Ejemplo

En este apartado, vamos a desarrollar un ejemplo de uso del programa, realizando algunas simulaciones básicas. Los pasos que vamos a dar son los siguientes:

- En primer lugar, vamos a abrir el modelo en blanco (archivo current.mdl) seleccionando el menú File->New->Model.

- Ahora, si no tenemos un modelo ya preparado, podemos ir copiando y pegando bloques de librería en el molde (current.mdl) para obtener el modelo que deseemos simular. **Es muy importante resaltar que, cuando queramos almacenar lo que hayamos hecho sobre current.mdl, hemos de guardarlo con otro nombre para no perder el molde para usos posteriores.** Si tenemos un modelo aparte, otra opción es copiar ese modelo y pegarlo en el molde. **Del mismo modo, habrá que almacenarlo con otro nombre (en el ejemplo, usaremos esta opción, dando al archivo el nombre currfingain.mdl).** Estos 2 pasos que se han descrito hasta ahora son de preparación, si esto ya se ha realizado antes podemos saltárnoslos y pasar directamente al tercer paso.

- Ahora, abriremos el modelo almacenado (en este ejemplo, currfingain.mdl) seleccionando el menú File->Open->Model. El resultado se observa en la figura A1.23., en la que puede verse cómo en el cuadro de texto Model Simulated de la Zona de Simulación aparece la ruta completa del archivo currfingain.mdl.

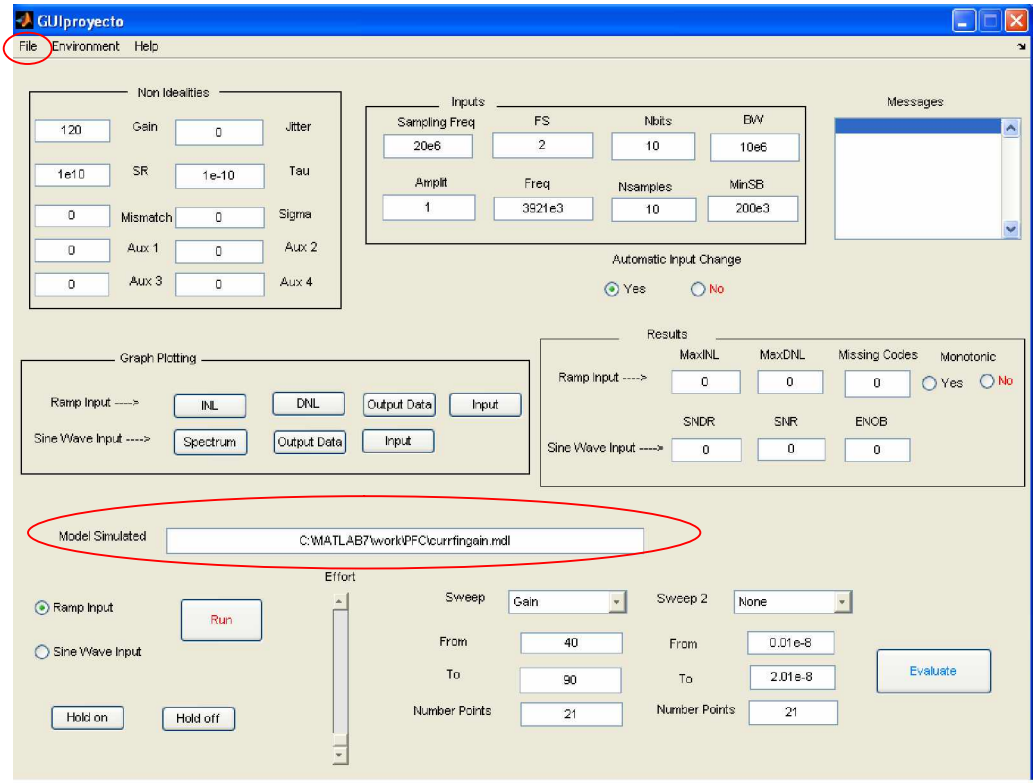


Figura A1.23.: Resultado de abrir currfingain.mdl

- Una vez efectuado el paso anterior, ya podemos hacer simulaciones. Si hacemos clic en la flecha superior de la barra Effort 4 veces (para

poner un tiempo intermedio de simulación para el caso de entrada rampa), y posteriormente pulsamos el botón Run de la Zona de Simulación, haremos una simulación simple del modelo currfingain.mdl con entrada rampa. En la zona de Results, aparecen diversos resultados de esta simulación (máxima INL, máxima DNL, códigos perdidos, y si la salida es monotónica o no). En la zona de mensajes, aparecerá primero un mensaje de simulación iniciada y después otro de simulación terminada. Además, pueden pulsarse los botones INL, DNL, Output Data e Input de la Zona de Graph Plotting para ver las gráficas que se deseen. El aspecto de la GUI al realizar esta simulación puede verse en la figura A1.24.

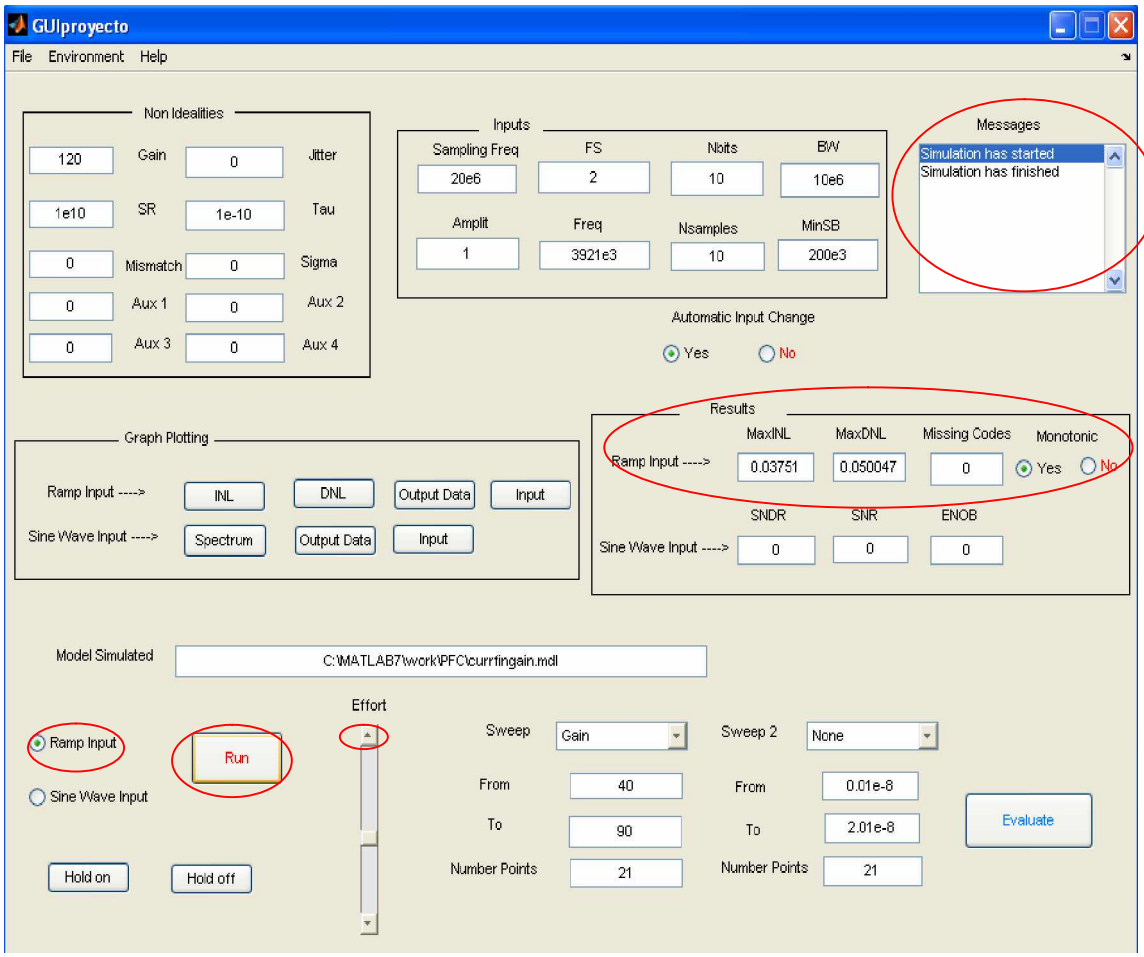


Figura A1.24.: GUI tras hacer simulación simple con entrada rampa

- Podríamos realizar una simulación simple con entrada seno haciendo clic en el radiobutton Sine Wave Input, haciendo clic en la flecha superior de la barra Effort 2 veces (para poner un tiempo intermedio de simulación para el caso de entrada seno), y después pulsando el botón Run. En la zona de Results, aparecen diversos resultados de esta simulación (SNDR, SNR y ENOB). En la zona de mensajes, aparecerá primero un mensaje de simulación iniciada y después otro de simulación terminada. Además, pueden pulsarse los botones Spectrum, Output Data e Input de la Zona de Graph Plotting para ver las gráficas que se deseen. El aspecto de la GUI sería el de la figura A1.25.

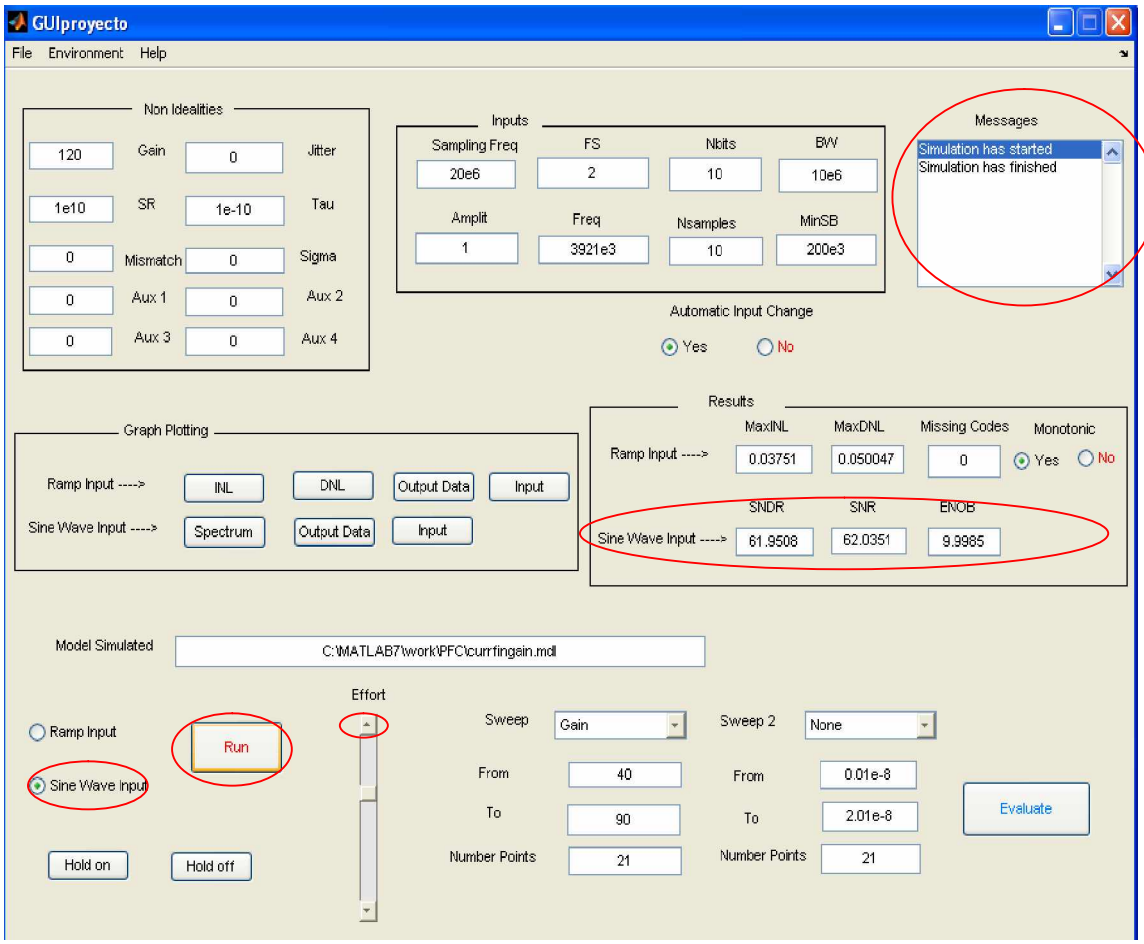


Figura A1.25.: GUI tras hacer simulación simple con entrada seno

- Ahora pasaremos a los barridos. Por ejemplo, haremos clic en el radiobutton Ramp Input, hacemos clic en la flecha inferior de la barra

Effort 2 veces (tiempo intermedio para rampa) y después pulsaremos el botón Evaluate. Con los valores de los pop-up menus seleccionados, esto provocará que se realice un barrido simple (en Sweep 2 está seleccionado None), de Ganancia Finita, entre 40 dB y 90 dB, de 21 puntos. En la zona de mensajes, aparecerá primero un mensaje de barrido de Ganancia Finita iniciado, después otro de barrido en proceso y al terminar otro de barrido finalizado. Al ser un barrido simple con entrada rampa, nos aparecerá al concluir el barrido una gráfica de INL máximo frente a la ganancia del amplificador en dB. El aspecto de la GUI tras el barrido se aprecia en la figura A1.26.

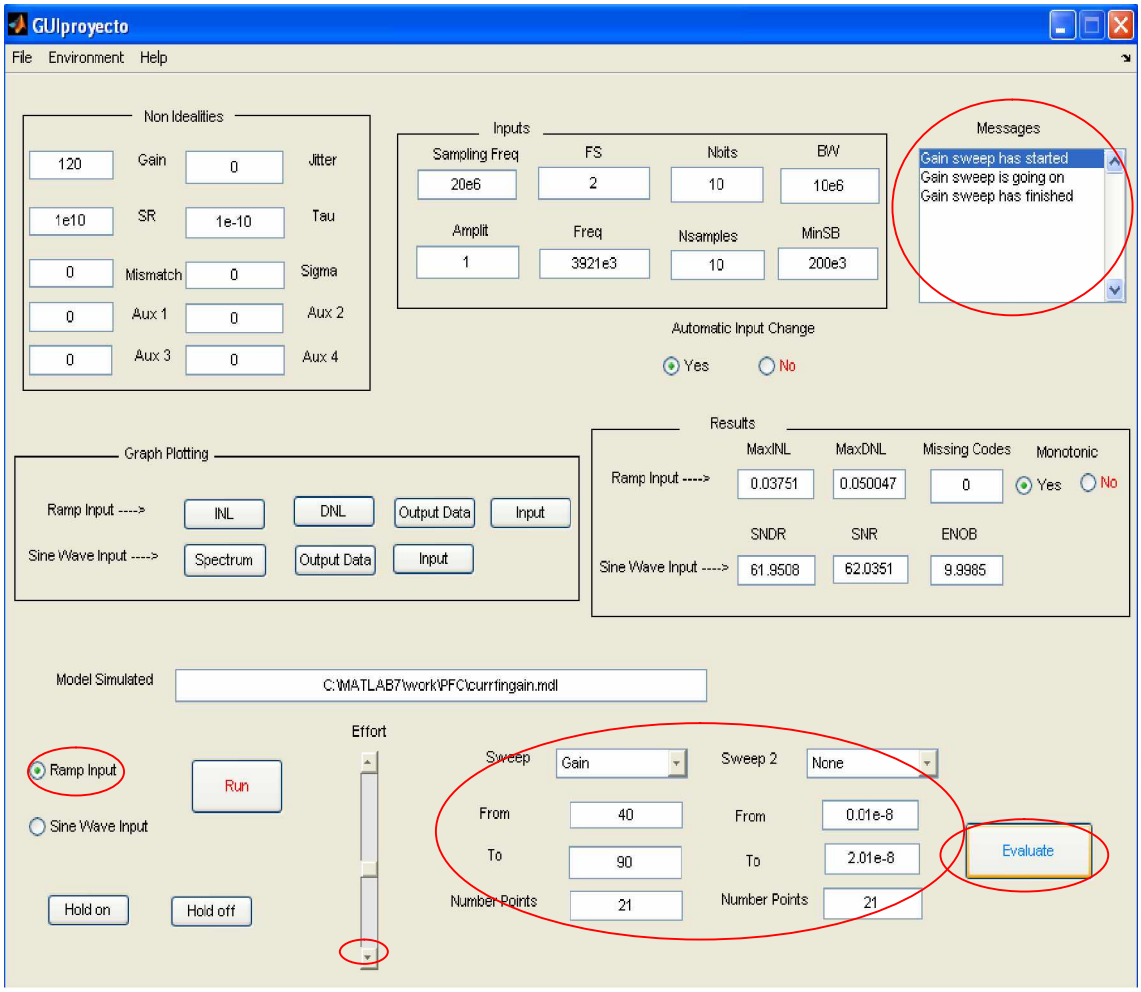


Figura A1.26.: GUI tras barrido simple de Ganancia Finita con entrada rampa

- Por último, terminaremos este ejemplo haciendo un barrido doble de Slew Rate y Tau. Para ello, cogeremos un modelo apropiado, es decir, que incluya estas 2 no idealidades, lo copiaremos y lo pegaremos en el

molde, guardando el archivo resultante como currslewratetau.mdl. Abrimos este archivo usando File->Menu->Open, después seleccionamos entrada seno, hacemos clic en la flecha superior de la barra Effort 2 veces (tiempo intermedio para seno), en Sweep elegimos SR (con From=1e7, To=2e7, Number Points=11) y en Sweep 2 elegimos Tau (con From=1e-10, To=1e-8, Number Points=11). Pulsamos el botón Evaluate. En la zona de mensajes, aparecerá primero un mensaje de barrido de Slew Rate y Tau iniciado, después otro de barrido en proceso y al terminar otro de barrido finalizado.Al terminar el barrido, nos aparecerá una gráfica en 3 dimensiones de SNDR en dB frente a Slew Rate y $\frac{1}{2\pi \cdot Tau}$. El aspecto de la GUI tras el barrido se aprecia en la figura A1.27.

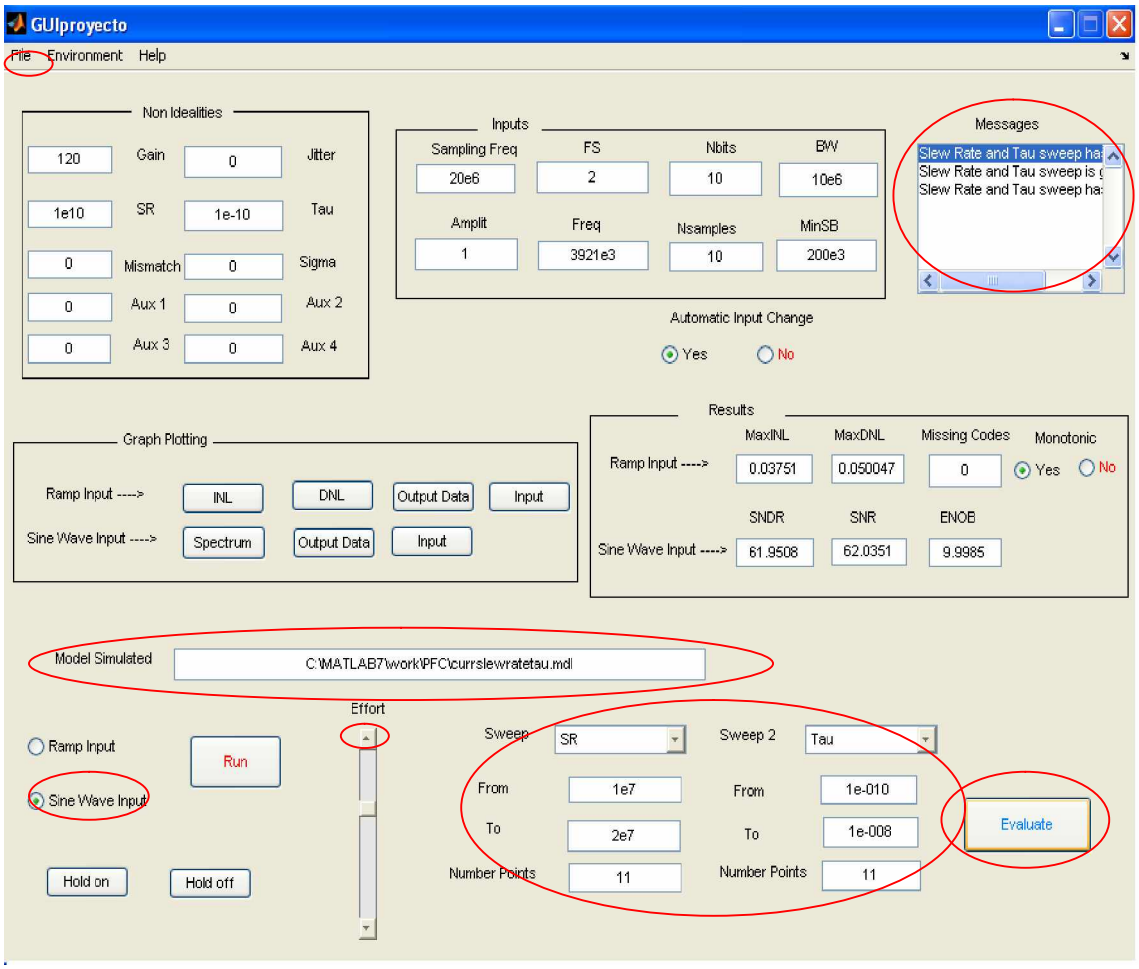


Figura A1.27.: GUI tras barrido doble de Slew Rate y Tau con entrada seno

Apéndice 2
Código Matlab

Este apéndice contiene el listado del código Matlab comentado del programa, se separarán los trozos de código por ficheros:

- Fichero simconverter.m:

```
% Script that runs the converter simulator

initial;          % Script that initializes program environment
                  % (global variables, path, etc.)
GUIproyecto;      % Script that has got the code that controls the
                  % behaviour of the GUI
```

- Fichero initial.m:

```
% Script that initializes program environment (global variables,
% path, etc.)

clear;
path(path,pwd);              % Add working directory to the path
cd Library;
path(path,pwd);              % Add Library directory to the path
cd ..;
global Ts;                   % Declare all the global variables needed
global FS;
global Eps;
global SR;
global Tau;
global Aux1;
global Aux2;
global Aux3;
global Aux4;
```

```
global INLMAX;
global DNLMAX;
global misingcode;
global Select;
global A;
global fsin;
global N;
global Ao;
global AodB;
global val;
global val2;
global vector;
global INLCOD;
global DNLCOD;
global Sndr;
global Snr;
global Enob;
global vectorcode;
global sigma;
global bandwidth;
global minsignalband;
global modelsimulated;
global VoffsetVrefpn;
global Voffset0;
global vectsndr;
global nsamples;
global siginput;
global axisxlabelVolt;
global axisxlabelTime;
global axisxlabelFrequency;
global axisylabelINL;
global axisylabelDNL;
global axisylabelOutput;
global axisylabelInput;
global axisylabelSpectrum;
global omegapole;
global maxINL;
global vectsndr3d;
global S;
global T;
global sigmajitter;
global Start;
global Stop;
global Points;
global Start2;
global Stop2;
global Points2;
global Effort;
global Auto;
```

- Fichero GUIproyecto.m:

```
% Script that has got the code that controls the behaviour of the GUI

function varargout = GUIproyecto(varargin)
% GUIPROYECTO M-file for GUIproyecto.fig
% GUIPROYECTO, by itself, creates a new GUIPROYECTO or raises the
% existing singleton*.
%
% H = GUIPROYECTO returns the handle to a new GUIPROYECTO or the
% handle to the existing singleton*.
%
% GUIPROYECTO('CALLBACK',hObject,eventData,handles,...) calls the
% local function named CALLBACK in GUIPROYECTO.M with the given input
% arguments.
%
% GUIPROYECTO('Property','Value',...) creates a new GUIPROYECTO or
% raises the existing singleton*.  Starting from the left, property
% value pairs are applied to the GUI before
% GUIproyecto_OpeningFunction gets called.  An unrecognized property
% name or invalid value makes property application stop.  All inputs
% are passed to GUIproyecto_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUIproyecto
% Last Modified by GUIDE v2.5 09-Jul-2005 17:11:11

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUIproyecto_OpeningFcn, ...
                   'gui_OutputFcn',  @GUIproyecto_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before GUIproyecto is made visible.
function GUIproyecto_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUIproyecto (see VARARGIN)

fig = handles.Current;              % Handle to the figure
```

```
GUIinit(handles);              % Iniatilize values of global variables

% Choose default command line output for GUIproyecto
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function   varargout   =   GUIproyecto_OutputFcn(hObject,   eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Initialize values of global variables
function GUIinit(handles)

global Ts;                        % Global variables to be initialized
global FS;
global A;
global fsin;
global N;
global sigma;
global sigmajitter;
global bandwidth;
global minsignalband;
global val;
global val2;
global Select;
global modelsimulated;
global VoffsetVrefpn;
global Voffset0;
global nsamples;
global Effort;
global Eps;
global SR;
global Tau;
global Aux1;
global Aux2;
global Aux3;
global Aux4;
global Ao;
global AodB;
global Auto;

% Initializing global variables from text boxes, etc.
Ts = 1/(str2double(get(handles.SamplingFrequency,'String')));
FS = str2double(get(handles.FullScale,'String'));
A = str2double(get(handles.Amplitude,'String'));
fsin = str2double(get(handles.Frequency,'String'));
N = str2double(get(handles.NumberBits,'String'));
sigma = str2double(get(handles.Sigma,'String'));   % Sigma is the
                                                   % typical deviation
                                                   % of the offset
                                                   % distribution
```

```
bandwidth = str2double(get(handles.Bandwidth,'String'));
minsignalband = str2double(get(handles.Minsignalband,'String'));
sigmajitter = str2double(get(handles.Jitter,'String'));
modelsimulated = get(handles.ModelSimulated,'String');
set(handles.ModelSimulated,'String',modelsimulated);

val = get(handles.Sweep_Popupmenu,'Value');
val2 = get(handles.Sweep2_Popupmenu,'Value');

slider_step(1)=1/9;
slider_step(2)=1/9;
set(handles.Effort_Slider,'sliderstep',slider_step,'Max',10.0,'Min',
1.0,'Value',1.0);
Effort = get(handles.Effort_Slider,'Value');


nsamples = str2double(get(handles.Nsamples,'String'));
VoffsetVrefpn = sigma * randn(1);       % Offsets are random numbers
                                        % with typical deviation sigma
Voffset0 = sigma * randn(1);
Select = 1;                             % Ramp Input by default

AodB = str2double(get(handles.Gain,'String'));
Eps = str2double(get(handles.Mismatch,'String'));
SR = str2double(get(handles.SR,'String'));
Tau = str2double(get(handles.Tau,'String'));
Aux1 = str2double(get(handles.Aux1,'String'));
Aux2 = str2double(get(handles.Aux2,'String'));
Aux3 = str2double(get(handles.Aux3,'String'));
Aux4 = str2double(get(handles.Aux4,'String'));
Ao = 10^(AodB/20);
Auto=1;                           % Automatic inputs change by default



% --- Executes during object creation, after setting all properties.
function SamplingFrequency_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SamplingFrequency (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
              get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Sampling Freq text box
function SamplingFrequency_Callback(hObject, eventdata, handles)
% hObject    handle to SamplingFrequency (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of SamplingFrequency
% as text
```

```
% str2double(get(hObject,'String')) returns contents of
% SamplingFrequency as a double
global Ts;
global bandwidth;
global fsin;
global minsignalband;
global Auto;
Ts = 1/(str2double(get(hObject,'String')))

% If automatic inputs change is selected, some values change
if(Auto==1)
    bandwidth = 1/(2*Ts);
    set(handles.Bandwidth,'String',bandwidth);
    fsin = 1/(5.1*Ts);
    set(handles.Frequency,'String',fsin);
    minsignalband = 1/(400*Ts);
    set(handles.Minsignalband,'String',minsignalband);
end

% --- Executes during object creation, after setting all properties.
function FullScale_CreateFcn(hObject, eventdata, handles)
% hObject    handle to FullScale (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing FS text box
function FullScale_Callback(hObject, eventdata, handles)
% hObject    handle to FullScale (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of FullScale as text
% str2double(get(hObject,'String')) returns contents of FullScale as a
% double
global FS;
global A;
global Auto;
FS = str2double(get(hObject,'String'))

% If automatic inputs change is selected, some values change
if(Auto==1)
    A = FS/2;
    set(handles.Amplitude,'String',A);
end

% --- Executes on button press in DNL_pushbutton.
function DNL_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to DNL_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global DNLCOD;
global axisxlabelVolt;
global axisylabelDNL;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','DNL','Visible','on',...
      'NumberTitle','off','HandleVisibility','on',
       'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(DNLCOD(1,:), DNLCOD(2,:),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','DNL')
xlabel(axisxlabelVolt)
ylabel(axisylabelDNL)
grid(handles.PlotAxes,'on')


% --- Executes on button press in Spectrum_pushbutton.
function Spectrum_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Spectrum_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global F;
global potencia1;
global axisxlabelFrequency;
global axisylabelSpectrum;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Spectrum','Visible','on',...
      'NumberTitle','off','HandleVisibility','on',
            'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot the figure on logarithmic axes, giving name, xlabel,
% ylabel and grid
semilogx(F,potencia1,'Parent',handles.PlotAxes);
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Spectrum')
xlabel(axisxlabelFrequency)
ylabel(axisylabelSpectrum)
grid(handles.PlotAxes,'on')


% --- Executes on button press in INL_pushbutton.
function INL_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to INL_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global INLCOD;
global axisxlabelVolt;
global axisylabelINL;
```

```
% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','INL','Visible','on',...
      'NumberTitle','off','HandleVisibility','on',
           'IntegerHandle','off','Position',[10 40 500 400]);
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
            guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(INLCOD(1,:), INLCOD(2,:),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','INL')
xlabel(axisxlabelVolt)
ylabel(axisylabelINL)
grid(handles.PlotAxes,'on')

% --- Executes on button press in Output_Data_SWI_pushbutton.
function    Output_Data_SWI_pushbutton_Callback(hObject,    eventdata,
handles)
% hObject    handle to Output_Data_SWI_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global vector;
global axisxlabelTime;
global axisylabelOutput;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Output','Visible','on',...
      'NumberTitle','off','HandleVisibility','on',
             'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(vector(:,1),vector(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Output')
xlabel(axisxlabelTime)
ylabel(axisylabelOutput)
grid(handles.PlotAxes,'on')

% --- Executes during object creation, after setting all properties.
function MaxINL_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MaxINL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
          get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% Not used (output result)
function MaxINL_Callback(hObject, eventdata, handles)
% hObject    handle to MaxINL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of MaxINL as text
%  str2double(get(hObject,'String')) returns contents of MaxINL
% as a double


% --- Executes during object creation, after setting all properties.
function MaxDNL_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MaxDNL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
       get(0,'defaultUicontrolBackgroundColor'));
end


% Not used (output result)
function MaxDNL_Callback(hObject, eventdata, handles)
% hObject    handle to MaxDNL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of MaxDNL as text
% str2double(get(hObject,'String')) returns contents of MaxDNL
% as a double


% --- Executes during object creation, after setting all properties.
function SNDR_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SNDR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
        get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% Not used (output result)
function SNDR_Callback(hObject, eventdata, handles)
% hObject    handle to SNDR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of SNDR as text
% str2double(get(hObject,'String')) returns contents of SNDR
% as a double

% --- Executes during object creation, after setting all properties.
function SNR_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SNR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
        get(0,'defaultUicontrolBackgroundColor'));
end

% Not used (output result)
function SNR_Callback(hObject, eventdata, handles)
% hObject    handle to SNR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of SNR as text
% str2double(get(hObject,'String')) returns contents of SNR
% as a double


% --- Executes during object creation, after setting all properties.
function Messages_Listbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Messages_Listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
      get(0,'defaultUicontrolBackgroundColor'));
end

% Not used (output messages)
function Messages_Listbox_Callback(hObject, eventdata, handles)
% hObject    handle to Messages_Listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = get(hObject,'String') returns Messages_Listbox
% contents as cell array
%  contents{get(hObject,'Value')} returns selected item from
% Messages_Listbox

% --- Executes on button press in Evaluate_pushbutton.
function Evaluate_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Evaluate_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Start;
global Stop;
global Points;
global Start2;
global Stop2;
global Points2;
global val;
global val2;

% Take sweep parameters from text boxes
Start = str2double(get(handles.Start,'String'))
Stop = str2double(get(handles.Stop,'String'))
Points = str2double(get(handles.Points,'String'))
Start2 = str2double(get(handles.Start2,'String'))
Stop2 = str2double(get(handles.Stop2,'String'))
Points2 = str2double(get(handles.Points2,'String'))

% Sweep select (if val2 equals 1 => Single Sweep;
% Others => Double sweep)
switch val2
    case 1                    % Sweep 2 = None => Single Sweep
        switch val
            case 1            % Gain sweep
                EstudioGananciaParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 2            % Mismatch sweep
                EstudioMismatchEtapa1ParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 3            % Slew Rate sweep
                EstudioSREtapa1ParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 4            % Tau sweep
                EstudioTauEtapa1ParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 5            % Sigma sweep
                EstudioSigmaParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 6            % Jitter sweep
                EstudioJitterParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 7            % Sampling Frequency sweep
                EstudioSamplingFreqParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 8            % Input amplitude sweep
                EstudioAmplitParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 9            % Full Scale sweep
                EstudioFSParamGlobal(Start,Stop,
                         Points,handles,hObject);
```

```
            case 10           % Input frequency sweep
                EstudioFreqParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 11           % Bandwidth sweep
                EstudioBWParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 12           % Aux 1 sweep
                EstudioAux1ParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 13           % Aux 2 sweep
                EstudioAux2ParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 14           % Aux 3 sweep
                EstudioAux3ParamGlobal(Start,Stop,
                          Points,handles,hObject);
            case 15           % Aux 4 sweep
                EstudioAux4ParamGlobal(Start,Stop,
                          Points,handles,hObject);
        end
    case 3                    % Sweep 2 = Mismatch
        switch val
            case 1            % Gain and Mismatch sweep
          EstudioGananciaMismatchParamGlobal(Start,Stop,Points,
                          Start2,Stop2,Points2,handles,hObject);
        end
    case 5                    % Sweep 2 = Tau
        switch val
            case 3            % Slew Rate and Tau sweep
                EstudioSRTauEtapa1ParamGlobal(Start,Stop,Points,
                          Start2,Stop2,Points2,handles,hObject);
        end
    case 6                    % Sweep 2 = Sigma
        switch val
            case 1            % Gain and Sigma sweep
                EstudioGananciaSigmaParamGlobal(Start,Stop,Points,
                          Start2,Stop2,Points2,handles,hObject);
            case 2            % Mismatch and Sigma sweep
                EstudioMismatchSigmaParamGlobal(Start,Stop,Points,
                          Start2,Stop2,Points2,handles,hObject);
            case 3            % Slew Rate and Sigma sweep
                EstudioSRSigmaParamGlobal(Start,Stop,Points,
                          Start2,Stop2,Points2,handles,hObject);
            case 4            % Tau and Sigma sweep
                EstudioTauSigmaParamGlobal(Start,Stop,Points,
                          Start2,Stop2,Points2,handles,hObject);
        end
    case 7                    % Sweep 2 = Jitter
        switch val
            case 1            % Gain and Jitter sweep
                EstudioGananciaJitterParamGlobal(Start,Stop,Points,
                           Start2,Stop2,Points2,handles,hObject);
            case 2            % Mismatch and Jitter sweep
                EstudioMismatchJitterParamGlobal(Start,Stop,Points,
                           Start2,Stop2,Points2,handles,hObject);


            case 3            % Slew Rate and Jitter sweep
                EstudioSRJitterParamGlobal(Start,Stop,Points,
                           Start2,Stop2,Points2,handles,hObject);
```

```
            case 4              % Tau and Jitter sweep
                EstudioTauJitterParamGlobal(Start,Stop,Points,
                            Start2,Stop2,Points2,handles,hObject);
            case 5              % Sigma and Jitter sweep
                EstudioSigmaJitterParamGlobal(Start,Stop,Points,
                            Start2,Stop2,Points2,handles,hObject);
        end
    case 8                      % Sweep 2 = Sampling Freq
        switch val
            case 1              % Gain and Sampling Frequency sweep
                EstudioGananciaSamplingFreqParamGlobal(Start,Stop,
                      Points,Start2,Stop2,Points2,handles,hObject);
            case 2              % Mismatch and Sampling Frequency sweep
                EstudioMismatchSamplingFreqParamGlobal(Start,Stop,
                      Points,Start2,Stop2,Points2,handles,hObject);
        end
    case 9                      % Sweep 2 = Amplit
        switch val
            case 1              % Gain and input amplitude sweep
                EstudioGananciaAmplitParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
            case 2              % Mismatch and input amplitude sweep
                EstudioMismatchAmplitParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
        end
    case 10                     % Sweep 2 = FS
        switch val
            case 1              % Gain and Full Scale sweep
                EstudioGananciaFSParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
            case 2              % Mismatch and Full Scale sweep
                EstudioMismatchFSParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
        end
    case 11                     % Sweep 2 = Freq
        switch val
            case 1              % Gain and input frequency sweep
                EstudioGananciaFreqParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
            case 2              % Mismatch and input frequency sweep
                EstudioMismatchFreqParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
        end
    case 14                     % Sweep 2 = Aux 2
        switch val
            case 12             % Aux 1 and Aux 2 sweep
                EstudioAux1Aux2ParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
        end
    case 15                     % Sweep 2 = Aux 3
        switch val
            case 12             % Aux 1 and Aux 3 sweep
                EstudioAux1Aux3ParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
            case 13             % Aux 2 and Aux 3 sweep
                EstudioAux2Aux3ParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject);
        end
```

```
    case 16                   % Sweep 2 = Aux 4
        switch val
            case 12           % Aux 1 and Aux 4 sweep
                EstudioAux1Aux4ParamGlobal(Start,Stop,Points,
                        Start2,Stop2,Points2,handles,hObject);
            case 13           % Aux 2 and Aux 4 sweep
                EstudioAux2Aux4ParamGlobal(Start,Stop,Points,
                        Start2,Stop2,Points2,handles,hObject);
            case 14           % Aux 3 and Aux 4 sweep
                EstudioAux3Aux4ParamGlobal(Start,Stop,Points,
                        Start2,Stop2,Points2,handles,hObject);
        end
end


% --- Executes during object creation, after setting all properties.
function Gain_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Gain text box
function Gain_Callback(hObject, eventdata, handles)
% hObject    handle to Gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Gain as text
%  str2double(get(hObject,'String')) returns contents of Gain
% as a double
global AodB;
global Ao;
AodB = str2double(get(hObject,'String'))
Ao = 10^(AodB/20)

% --- Executes during object creation, after setting all properties.
function SR_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%  See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
        get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% --- Executes when editing SR text box
function SR_Callback(hObject, eventdata, handles)
% hObject    handle to SR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of SR as text
%  str2double(get(hObject,'String')) returns contents of SR
% as a double
global SR;
SR = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Mismatch_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Mismatch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Mismatch text box
function Mismatch_Callback(hObject, eventdata, handles)
% hObject    handle to Mismatch (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Mismatch as text
% str2double(get(hObject,'String')) returns contents of Mismatch
% as a double
global Mismatch;
Mismatch = str2double(get(hObject,'String'))


% --- Executes on button press in Run_pushbutton.
function Run_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Run_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Gain;
global Mismatch;
global SlewRate;
global TimeConstant;
global Aux1;
global Aux2;
global Aux3;
global Aux4;
global sigmajitter;
global Select;
global INLMAX;
global DNLMAX;
global misingcode;
global Sndr;
```

```
global Snr;
global Enob;
global axisxlabel;
global axisylabel;

% Take single simulation parameters from text boxes
Gain = str2double(get(handles.Gain,'String'));
Mismatch = str2double(get(handles.Mismatch,'String'));
SlewRate = str2double(get(handles.SR,'String'));
TimeConstant = str2double(get(handles.Tau,'String'));
Aux1 = str2double(get(handles.Aux1,'String'));
Aux2 = str2double(get(handles.Aux2,'String'));
Aux3 = str2double(get(handles.Aux3,'String'));
Aux4 = str2double(get(handles.Aux4,'String'));
sigmajitter = str2double(get(handles.Jitter,'String'));
sigma = str2double(get(handles.Sigma,'String'));

% Offsets are random numbers with typical deviation sigma
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);

% Calling to the function which controls single simulations
SimulacionNoIdeal(Gain,Mismatch,SlewRate,TimeConstant,
        sigmajitter,sigma,Aux1,Aux2,Aux3,Aux4,handles,hObject);

% Which results change depends on the input type
if (Select==1)     % Select==1 equals Ramp Input => Change INL, DNL
                   % and MissingCodes
         set(handles.MaxINL,'String',num2str(INLMAX));
         set(handles.MaxDNL,'String',num2str(DNLMAX));
         set(handles.MissingCodes,'String',num2str(misingcode));
else             % Select==0 equals Sine Wave Input => Change SNDR,
                 % SNR and ENOB
         set(handles.SNDR,'String',num2str(Sndr));
         set(handles.SNR,'String',num2str(Snr));
         set(handles.ENOB,'String',num2str(Enob));
end

% --- Executes during object creation, after setting all properties.
function Start_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
              get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Start text box
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of Start as text
% str2double(get(hObject,'String')) returns contents of Start
% as a double
global Start;
Start = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Stop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Stop text box
function Stop_Callback(hObject, eventdata, handles)
% hObject    handle to Stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Stop as text
%  str2double(get(hObject,'String')) returns contents of Stop
% as a double
global Stop;
Stop = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Points_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Points (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing Points text box
function Points_Callback(hObject, eventdata, handles)
% hObject    handle to Points (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of Points as text
% str2double(get(hObject,'String')) returns contents of Points
% as a double
global Points;
Points = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Tau_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Tau (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
           get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Tau text box
function Tau_Callback(hObject, eventdata, handles)
% hObject    handle to Tau (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Tau as text
% str2double(get(hObject,'String')) returns contents of Tau
% as a double
global Tau;
Tau = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Start2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Start2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
          get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing Start2 text box
function Start2_Callback(hObject, eventdata, handles)
% hObject    handle to Start2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Start2 as text
% str2double(get(hObject,'String')) returns contents of Start2
% as a double
global Start2;
Start2 = str2double(get(hObject,'String'))
```

```
% --- Executes during object creation, after setting all properties.
function Stop2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Stop2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
           get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Stop2 text box
function Stop2_Callback(hObject, eventdata, handles)
% hObject    handle to Stop2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Stop2 as text
% str2double(get(hObject,'String')) returns contents of Stop2
% as a double
global Stop2;
Stop2 = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Points2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Points2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
       get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Points2 text box
function Points2_Callback(hObject, eventdata, handles)
% hObject    handle to Points2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Points2 as text
% str2double(get(hObject,'String')) returns contents of Points2
% as a double
global Points2;
Points2 = str2double(get(hObject,'String'))
```

```
% --- Executes during object creation, after setting all properties.
function Amplitude_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Amplitude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing A text box
function Amplitude_Callback(hObject, eventdata, handles)
% hObject    handle to Amplitude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Amplitude as text
%  str2double(get(hObject,'String')) returns contents of Amplitude
% as a double
global A;
A = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Frequency_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Frequency (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
           get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing Freq text box
function Frequency_Callback(hObject, eventdata, handles)
% hObject    handle to Frequency (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Frequency as text
%  str2double(get(hObject,'String')) returns contents of Frequency
% as a double
global fsin;
fsin = str2double(get(hObject,'String'))
```

```
% --- Executes during object creation, after setting all properties.
function NumberBits_CreateFcn(hObject, eventdata, handles)
% hObject    handle to NumberBits (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing N text box
function NumberBits_Callback(hObject, eventdata, handles)
% hObject    handle to NumberBits (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of NumberBits as text
% str2double(get(hObject,'String')) returns contents of NumberBits
% as a double
global N;
N = str2double(get(hObject,'String'))


% --- Executes on button press in SineWaveInput_Radiobutton.
function     SineWaveInput_Radiobutton_Callback(hObject,     eventdata,
handles)
% hObject    handle to SineWaveInput_Radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of
% SineWaveInput_Radiobutton

% If this button turns on, RampInput will be turned off
off = [handles.RampInput_Radiobutton];
mutual_exclude(off)
global Select;
Select=0;                    % Sine Wave Input


% --- Executes on button press in RampInput_Radiobutton.
function RampInput_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to RampInput_Radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of
% RampInput_Radiobutton

% If this button turns on, SineWaveInput will be turned off
off= [handles.SineWaveInput_Radiobutton];
mutual_exclude(off)
global Select;
Select=1;                     % Ramp Input
```

```
% --- Function that ensures mutual exclusion between a group
%  of radiobuttons
function mutual_exclude(off)
set(off,'Value',0)

% --- Executes during object creation, after setting all properties.
function Jitter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Jitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Jitter text box
function Jitter_Callback(hObject, eventdata, handles)
% hObject    handle to Jitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Jitter as text
%  str2double(get(hObject,'String')) returns contents of Jitter
% as a double
global sigmajitter;
sigmajitter = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function MissingCodes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MissingCodes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
            get(0,'defaultUicontrolBackgroundColor'));
end

% Not used (output result)
function MissingCodes_Callback(hObject, eventdata, handles)
% hObject    handle to MissingCodes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of MissingCodes
% as text
% str2double(get(hObject,'String')) returns contents of MissingCodes
% as a double
```

```
% --- Executes during object creation, after setting all properties.
function Sigma_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sigma (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
           get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Sigma text box
function Sigma_Callback(hObject, eventdata, handles)
% hObject    handle to Sigma (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Sigma as text
%  str2double(get(hObject,'String')) returns contents of Sigma
% as a double
global sigma;
global VoffsetVrefpn;
global Voffset0;
sigma = str2double(get(hObject,'String'))
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);

% --- Executes on button press in Output_Data_RI_pushbutton.
function     Output_Data_RI_pushbutton_Callback(hObject,     eventdata,
handles)
% hObject    handle to Output_Data_RI_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global vector;
global vectorcode;
global axisxlabelVolt;
global axisylabelOutput;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Output','Visible','on',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(vectorcode(:,1),vector(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Output')
xlabel(axisxlabelVolt)
ylabel(axisylabelOutput)
grid(handles.PlotAxes,'on')
```

```
% --- Executes during object creation, after setting all properties.
function ENOB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ENOB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end


% Not used (output result)
function ENOB_Callback(hObject, eventdata, handles)
% hObject    handle to ENOB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ENOB as text
%  str2double(get(hObject,'String')) returns contents of ENOB
% as a double


% --- Executes during object creation, after setting all properties.
function Bandwidth_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Bandwidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing BW text box
function Bandwidth_Callback(hObject, eventdata, handles)
% hObject    handle to Bandwidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Bandwidth as text
% str2double(get(hObject,'String')) returns contents of Bandwidth
% as a double
global bandwidth;
bandwidth = str2double(get(hObject,'String'))
```

```
% --- Executes during object creation, after setting all properties.
function Minsignalband_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Minsignalband (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing MinSB text box
function Minsignalband_Callback(hObject, eventdata, handles)
% hObject    handle to Minsignalband (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Minsignalband
% as text
% str2double(get(hObject,'String')) returns contents of Minsignalband
% as a double
global minsignalband;
minsignalband = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function ModelSimulated_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ModelSimulated (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
          get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing Model Simulated text box
function ModelSimulated_Callback(hObject, eventdata, handles)
% hObject    handle to ModelSimulated (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ModelSimulated
% as text
% str2double(get(hObject,'String')) returns contents of ModelSimulated
% as a double
global modelsimulated;
modelsimulated = get(handles.ModelSimulated,'String')
```

```
% --- Executes during object creation, after setting all properties.
function Nsamples_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Nsamples (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
        get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing Nsamples text box
function Nsamples_Callback(hObject, eventdata, handles)
% hObject    handle to Nsamples (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Nsamples as text
% str2double(get(hObject,'String')) returns contents of Nsamples
% as a double
global nsamples;
nsamples = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Sweep2_Popupmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sweep2_Popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
        get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in Sweep2_Popupmenu.
function Sweep2_Popupmenu_Callback(hObject, eventdata, handles)
% hObject    handle to Sweep2_Popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns Sweep2_Popupmenu
% contents as cell array
% contents{get(hObject,'Value')} returns selected item
% from Sweep2_Popupmenu
global val2;
global Start2;
global Stop2;
global Points2;
val2 = get(hObject,'Value');
```

```
% Automatic values on Sweep 2 parameters depends on the option
% selected on Sweep 2
switch val2
    case 2                          % Sweep 2 = Gain
        Start2=40;
        Stop2=90;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 3                           % Sweep 2 = Mismatch
        Start2=-0.2;
        Stop2=0.2;
        Points2=41;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 4                           % Sweep 2 = SR
        Start2=14e8;
        Stop2=38e8;
        Points2=25;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 5                           % Sweep 2 = Tau
        Start2=0.01e-8;
        Stop2=2.01e-8;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 6                           % Sweep 2 = Sigma
        Start2=1e-3;
        Stop2=1e-2;
        Points2=11;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 7                           % Sweep 2 = Jitter
        Start2=1e-8;
        Stop2=1e-12;
        Points2=41;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 8                           % Sweep 2 = Sampling Freq
        Start2=2e8;
        Stop2=2e6;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 9                           % Sweep 2 = Amplit
        Start2=0.1;
        Stop2=1;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
```

```
    case 10                          % Sweep 2 = FS
        Start2=1;
        Stop2=3;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 11                          % Sweep 2 = Freq
        Start2=0.5e6;
        Stop2=4e6;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 12                           % Sweep 2 = BW
        Start2=5e6;
        Stop2=15e6;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 13                           % Sweep 2 = Aux 1
        Start2=1e-2;
        Stop2=1e-3;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
     case 14                          % Sweep 2 = Aux 2
        Start2=1e-2;
        Stop2=1e-3;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 15                          % Sweep 2 = Aux 3
        Start2=1e-2;
        Stop2=1e-3;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
    case 16                          % Sweep 2 = Aux 4
        Start2=1e-2;
        Stop2=1e-3;
        Points2=21;
        set(handles.Start2,'String',Start2);
        set(handles.Stop2,'String',Stop2);
        set(handles.Points2,'String',Points2);
end

% --- Executes during object creation, after setting all properties.
function Aux1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
          get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Aux 1 text box
function Aux1_Callback(hObject, eventdata, handles)
% hObject    handle to Aux1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Aux1 as text
% str2double(get(hObject,'String')) returns contents of Aux1
% as a double
global Aux1;
Aux1 = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Aux3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
           get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes when editing Aux 3 text box
function Aux3_Callback(hObject, eventdata, handles)
% hObject    handle to Aux3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Aux3 as text
% str2double(get(hObject,'String')) returns contents of Aux3
% as a double
global Aux3;
Aux3 = str2double(get(hObject,'String'))

% --- Executes during object creation, after setting all properties.
function Aux2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
        get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% --- Executes when editing Aux 2 text box
function Aux2_Callback(hObject, eventdata, handles)
% hObject    handle to Aux2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Aux2 as text
% str2double(get(hObject,'String')) returns contents of Aux2
% as a double
global Aux2;
Aux2 = str2double(get(hObject,'String'))


% --- Executes during object creation, after setting all properties.
function Aux4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Aux4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
         get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes when editing Aux 4 text box
function Aux4_Callback(hObject, eventdata, handles)
% hObject    handle to Aux4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Aux4 as text
% str2double(get(hObject,'String')) returns contents of Aux4
% as a double
global Aux4;
Aux4 = str2double(get(hObject,'String'))


% --- Executes on button press in RampInput_pushbutton.
function RampInput_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to RampInput_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global siginput;
global axisxlabelTime;
global axisylabelInput;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Input','Visible','on',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end
```

```
% Plot the figure, giving name, xlabel, ylabel and grid
plot(siginput(:,1),siginput(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Input')
xlabel(axisxlabelTime)
ylabel(axisylabelInput)
grid(handles.PlotAxes,'on')

% --- Executes on button press in SineWaveInput_pushbutton.
function     SineWaveInput_pushbutton_Callback(hObject,     eventdata,
handles)
% hObject    handle to SineWaveInput_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global siginput;
global axisxlabelTime;
global axisylabelInput;

% If figure does not exist, create it
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Input','Visible','on',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot the figure, giving name, xlabel, ylabel and grid
plot(siginput(:,1),siginput(:,2),'Parent',handles.PlotAxes)
set(0,'CurrentFigure',handles.PlotFigure)
set(handles.PlotFigure,'Name','Input')
xlabel(axisxlabelTime)
ylabel(axisylabelInput)
grid(handles.PlotAxes,'on')

% Not used (output result)
function     Yes_Monotonic_Radiobutton_Callback(hObject,     eventdata,
handles)
% hObject    handle to Yes_Monotonic_Radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of
% Yes_Monotonic_Radiobutton

% Not used (output result)
function     No_Monotonic_Radiobutton_Callback(hObject,     eventdata,
handles)
% hObject    handle to No_Monotonic_Radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of
% No_Monotonic_Radiobutton


% --- Executes on button press in HoldOn_pushbutton.
function HoldOn_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to HoldOn_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% If figure exists, set it current and hold on graphics
if isfield(handles,'PlotFigure') & ishandle(handles.PlotFigure),
    set(0,'CurrentFigure',handles.PlotFigure);
    hold on;
end


% --- Executes on button press in HoldOff_pushbutton.
function HoldOff_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to HoldOff_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% If figure exists, set it current and hold off graphics
if isfield(handles,'PlotFigure') & ishandle(handles.PlotFigure),
    set(0,'CurrentFigure',handles.PlotFigure);
    hold off;
end


% --- Executes during object creation, after setting all properties.
function Sweep_Popupmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sweep_Popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',
          get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in Sweep_Popupmenu.
function Sweep_Popupmenu_Callback(hObject, eventdata, handles)
% hObject    handle to Sweep_Popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns Sweep_Popupmenu
% contents as cell array
% contents{get(hObject,'Value')} returns selected item
% from Sweep_Popupmenu
global val;
global Start;
global Stop;
global Points;
val = get(hObject,'Value');
```

```
% Automatic values on Sweep parameters depends on the option
% selected on Sweep
switch val
    case 1                      % Sweep = Gain
        Start=40;
        Stop=90;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 2                       % Sweep = Mismatch
        Start=-0.2;
        Stop=0.2;
        Points=41;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 3                       % Sweep = SR
        Start=14e8;
        Stop=38e8;
        Points=25;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 4                        % Sweep = Tau
        Start=0.01e-8;
        Stop=2.01e-8;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 5                        % Sweep = Sigma
        Start=1e-3;
        Stop=1e-2;
        Points=11;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 6                        % Sweep = Jitter
        Start=1e-8;
        Stop=1e-12;
        Points=41;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 7                         % Sweep = Sampling Freq
        Start=2e8;
        Stop=2e6;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 8                         % Sweep = Amplit
        Start=0.1;
        Stop=1;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
```

```
    case 9                          % Sweep = FS
        Start=1;
        Stop=3;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 10                         % Sweep = Freq
        Start=0.5e6;
        Stop=4e6;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 11                         % Sweep = BW
        Start=5e6;
        Stop=15e6;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 12                          % Sweep = Aux 1
        Start=1e-2;
        Stop=1e-3;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 13                           % Sweep = Aux 2
        Start=1e-2;
        Stop=1e-3;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 14                           % Sweep = Aux 3
        Start=1e-2;
        Stop=1e-3;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
    case 15                           % Sweep = Aux 4
        Start=1e-2;
        Stop=1e-3;
        Points=21;
        set(handles.Start,'String',Start);
        set(handles.Stop,'String',Stop);
        set(handles.Points,'String',Points);
end


% Not used------------------------------------------------------------
function file_menu_Callback(hObject, eventdata, handles)
% hObject    handle to file_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Not used------------------------------------------------------------
function menu_file_new_Callback(hObject, eventdata, handles)
% hObject    handle to menu_file_new (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selecting menu File->New->Model
function file_new_model_Callback(hObject, eventdata, handles)
% hObject    handle to file_new_model (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global modelsimulated;

% If is not opened, open file "current.mdl"
if  isempty(find_system('Name','current')),
    open_system('current')
    figure(handles.Current)
end

% Write on Model Simulated text box the complete route
% of "current.mdl"
modelsimulated = [];
modelsimulated = strcat(pwd,'\current.mdl');

set(handles.ModelSimulated,'String',modelsimulated);


% --- Executes on selecting menu File->Open->Model
function file_open_model_Callback(hObject, eventdata, handles)
% hObject    handle to file_open_model (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global modelsimulated;

% Open a dialog box for selecting the file to be opened
[filename, pathname] = uigetfile( ...
    {'*.mdl', 'All Model and Library Files (*.mdl)'; ...
        '*.*','All Files (*.*)'}, ...
    'Select Model');

% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
    return
% Otherwise construct the fullfilename and Check and load the file.
else
    File = fullfile(pathname,filename);

    handles.LastFIle = File;
    guidata(hObject,handles)
    open_system(File);

% Write on Model Simulated text box the complete route of the file
    modelsimulated = [];
    modelsimulated = File;
    set(handles.ModelSimulated,'String',modelsimulated);
end
```

```
% --- Executes on selecting menu File->Open->Library
function file_open_library_Callback(hObject, eventdata, handles)
% hObject    handle to file_open_library (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Add Library directory to the path
diract=pwd;
cd Library;
dirlib=pwd;
path(path,dirlib)

% Open a dialog box for selecting the file to be opened,
% starting in Library directory
[filename, pathname] = uigetfile( ...
    {'*.mdl', 'All Model and Library Files (*.mdl)'; ...
        '*.*','All Files (*.*)'}, ...
    'Select Library');
cd ..;
% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
    return
% Otherwise construct the fullfilename and Check and load the file.
else
    File = fullfile(pathname,filename);

    handles.LastFIle = File;
    guidata(hObject,handles)
    open_system(File);
end

% Not used-----------------------------------------------------------
function environment_menu_Callback(hObject, eventdata, handles)
% hObject    handle to environment_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selecting menu Environment->Load
function menu_environment_load_Callback(hObject, eventdata, handles)
% hObject    handle to menu_environment_load (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global modelsimulated;
global Ts;
global FS;
global N;
global bandwidth;
global minsignalband;
global fsin;
global A;
global nsamples;
global AodB;
global Mismatch;
global SR;
global Tau;
global sigma;
global sigmajitter;
global Aux1;
global Aux2;
```

```
global Aux3;
global Aux4;
global val;
global val2;
global Start;
global Stop;
global Points;
global Start2;
global Stop2;
global Points2;

diract=pwd;
cd States;

% Open a dialog box for selecting the file to be run,
% starting in States directory
[filename, pathname] = uigetfile( ...
    {'*.m','M-Files (*.m)'}, ...
    'Select Environment File');
cd ..;

% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
    return
% Otherwise construct the fullfilename and Check and run the file.
else
    File = fullfile(pathname,filename);
    handles.LastFIle = File;
    guidata(hObject,handles)
    run(File);
end

% --- Executes on selecting menu Environment->Save
function menu_environment_save_Callback(hObject, eventdata, handles)
% hObject    handle to menu_environment_save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

diract=pwd;
cd States;

% Open a dialog box for selecting the name of the file in which
% the actual state will be saved, starting in States directory
[filename, pathname] = uiputfile( ...
    {'*.m','M-Files (*.m)'}, ...
    'Save as');
cd ..;
% If "Cancel" is selected then return
if isequal([filename,pathname],[0,0])
    return
% Otherwise construct the fullfilename and Check and save the file.
else
    File = fullfile(pathname,filename);

    handles.LastFIle = File;
    guidata(hObject,handles)

% Open file with write permission
    fid=fopen(File,'w');
```

```
% Write all the code for the m-file that saves the code which
% will be run when "load" option would be selected, and so
% the actual state would be loaded
    fprintf(fid,'Ts=1/');
    fprintf(fid,get((handles.SamplingFrequency),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.SamplingFrequency,''String'',1/Ts)\n');
    fprintf(fid,'FS=');
    fprintf(fid,get((handles.FullScale),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.FullScale,''String'',FS)\n');
    fprintf(fid,'N=');
    fprintf(fid,get((handles.NumberBits),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.NumberBits,''String'',N)\n');
    fprintf(fid,'bandwidth=');
    fprintf(fid,get((handles.Bandwidth),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Bandwidth,''String'',bandwidth)\n');
    fprintf(fid,'A=');
    fprintf(fid,get((handles.Amplitude),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Amplitude,''String'',A)\n');
    fprintf(fid,'fsin=');
    fprintf(fid,get((handles.Frequency),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Frequency,''String'',fsin)\n');
    fprintf(fid,'nsamples=');
    fprintf(fid,get((handles.Nsamples),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Nsamples,''String'',nsamples)\n');
    fprintf(fid,'minsignalband=');
    fprintf(fid,get((handles.Minsignalband),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Minsignalband,
              ''String'',minsignalband)\n');
    fprintf(fid,'AodB=');
    fprintf(fid,get((handles.Gain),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Gain,''String'',AodB)\n');
    fprintf(fid,'Eps=');
    fprintf(fid,get((handles.Mismatch),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Mismatch,''String'',Eps)\n');
    fprintf(fid,'SR=');
    fprintf(fid,get((handles.SR),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.SR,''String'',SR)\n');
    fprintf(fid,'Tau=');
    fprintf(fid,get((handles.Tau),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Tau,''String'',Tau)\n');
    fprintf(fid,'sigmajitter=');
    fprintf(fid,get((handles.Jitter),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Jitter,''String'',sigmajitter)\n');
    fprintf(fid,'sigma=');
```

```
    fprintf(fid,get((handles.Sigma),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Sigma,''String'',sigma)\n');
    fprintf(fid,'Aux1=');
    fprintf(fid,get((handles.Aux1),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Aux1,''String'',Aux1)\n');
    fprintf(fid,'Aux2=');
    fprintf(fid,get((handles.Aux2),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Aux2,''String'',Aux2)\n');
    fprintf(fid,'Aux3=');
    fprintf(fid,get((handles.Aux3),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Aux3,''String'',Aux3)\n');
    fprintf(fid,'Aux4=');
    fprintf(fid,get((handles.Aux4),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Aux4,''String'',Aux4)\n');
    fprintf(fid,'val=');
    fprintf(fid,'%d',get((handles.Sweep_Popupmenu),'Value'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Sweep_Popupmenu,''Value'',val)\n');
    fprintf(fid,'val2=');
    fprintf(fid,'%d',get((handles.Sweep2_Popupmenu),'Value'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Sweep2_Popupmenu,''Value'',val2)\n');
    fprintf(fid,'Start=');
    fprintf(fid,get((handles.Start),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Start,''String'',Start)\n');
    fprintf(fid,'Stop=');
    fprintf(fid,get((handles.Stop),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Stop,''String'',Stop)\n');
    fprintf(fid,'Points=');
    fprintf(fid,get((handles.Points),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Points,''String'',Points)\n');
    fprintf(fid,'Start2=');
    fprintf(fid,get((handles.Start2),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Start2,''String'',Start2)\n');
    fprintf(fid,'Stop2=');
    fprintf(fid,get((handles.Stop2),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Stop2,''String'',Stop2)\n');
    fprintf(fid,'Points2=');
    fprintf(fid,get((handles.Points2),'String'));
    fprintf(fid,';\n');
    fprintf(fid,'set(handles.Points2,''String'',Points2)\n');

% Close file
    fclose(fid);
end
```

```
% Not used------------------------------------------------------------
function help_menu_Callback(hObject, eventdata, handles)
% hObject    handle to help_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selecting menu Help->User Guide
function menu_help_userguide_Callback(hObject, eventdata, handles)
% hObject    handle to menu_help_userguide (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Open User Guide (pdf format)
open('ManualdeUsuario.pdf')

% --- Executes on selecting menu Environment->Save---------------
function menu_help_about_Callback(hObject, eventdata, handles)
% hObject    handle to menu_help_about (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Create the character array for the msgbox
aboutmsg={'SimConverter ver 1.0','','Authors:','',
        'Daniel Falcon Medina','Fernando Munoz Chavero'};

% Display the msgbox, which will be modal
msgbox(aboutmsg,'About','modal')


% --- Executes during object creation, after setting all properties.
function Effort_Slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Effort_Slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
% CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else
    set(hObject,'BackgroundColor',
       get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function Effort_Slider_Callback(hObject, eventdata, handles)
% hObject    handle to Effort_Slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine range
% of slider
global Effort;
Effort=get(hObject,'Value')
```

```
% --- Executes on button press in Yes_Auto_Radiobutton.
function Yes_Auto_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to Yes_Auto_Radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% Yes_Auto_Radiobutton

% If this button turns on, No_Auto will be turned off
off= [handles.No_Auto_Radiobutton];
mutual_exclude(off)
global Auto;
Auto=1;                         % Yes_Auto

% --- Executes on button press in No_Auto_Radiobutton.
function No_Auto_Radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to No_Auto_Radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% No_Auto_Radiobutton

% If this button turns on, Yes_Auto will be turned off
off= [handles.Yes_Auto_Radiobutton];
mutual_exclude(off)
global Auto;
Auto=0;                                     % No_Auto

% Not used-----------------------------------------------------------
function menu_file_open_Callback(hObject, eventdata, handles)
% hObject    handle to menu_file_open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

- Fichero CalcSNDR.m:

```
% Function that evaluates SNDR, SNR and ENOB

function CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject)
global Ts;
global Sndr;
global Snr;
global Enob;
global fsin;
global F;
global potencia1;

fs=1/Ts;               % Sampling frequency
fb=fsin;               % Input signal frequency
fl=minsignalband;      % Lower band boundary
nb=nsamples;           % Signal bandwidth in samples.
                       % From center to a extreme.
bw=bandwidth;          % Bandwidth
```

```
tam=size(vector);
tam2=tam(1);
Npuntos_2 = floor(log2(tam2-12));     % Approximate to a lower integer
Npuntos=pow2(Npuntos_2);
res=vector(12:Npuntos+11,2);          % Take from vector matrix items
                                      % from 12 to Npuntos+11

fN=fs/2;                              % Nyquist frequency

Nfb=ceil(Npuntos*fb/fs);              % Number of the sample where is
                                      % fb, approximated to an upper
                                      % integer
Nbw=ceil(Npuntos*bw/fs);              % Bandwidth in samples
Nlo=ceil(Npuntos*fl/fs);              % Lower band boundary in samples
NfN=ceil(Npuntos*fN/fs);              % Nyquist frequency in samples
harmonics=floor(NfN/Nfb);

if (harmonics>15)
    harmonics=15;
end

% Evaluate PSD of discrete signal res, with Npuntos points,
% kaiser window and sampling frequency fs
[potencia,F]=psd(res,Npuntos,fs,kaiser(Npuntos,20));
potencia1=10*log10(potencia);                % Convert PSD to dB

senal=0;

% Sweep from extreme to extreme (Nfb is the central sampling,
% where is fb)
for i=Nfb-nb:Nfb+nb,
    senal=senal+potencia(i);
    potencia(i)=0;          % Put this to zero because it is not noise
end

% Evaluate noise
tampot=size(potencia);
tampot2=tampot(1);
if (Nbw<=tampot2)
    vruido=potencia(Nlo:Nbw);
else
    vruido=potencia(Nlo:tampot2);
end

% Evaluate SNDR
Sndr=10*log10(senal/sum(vruido))

for r=2:harmonics,
    potharm(r)=0;
    for g=((r*Nfb)-nb):((r*Nfb)+nb),
        if(g<((Npuntos/2)+2))
            potharm(r)=potharm(r)+potencia(g);
            potencia(g)=0;          % Put this to zero because
                                    % it is not distorsion
        end
    end
end
```

```
% Evaluate noise plus distorsion
if (Nbw<=tampot)
    vruidosindist=potencia(Nlo:Nbw);
else
    vruidosindist=potencia(Nlo:tampot2);
end

% Evaluate SNR
Snr=10*log10(senal/sum(vruidosindist))

% Evaluate ENOB
Enob=(Sndr-1.76)/6.02
```

- Fichero CalcINLDNL.m

```
% Function that evaluates INL, DNL and missing codes

function CalcINLDNL(b,handles)
global INLMAX;
global DNLMAX;
global INL;
global DNL;
global misingcode;

% Calling to the script that obtains INL, DNL and missing codes
INLDNL_matlab

INLAUX=abs(INL);
DNLAUX=abs(DNL);

INLMAX=max(INLAUX)+misingcode;
DNLMAX=max(DNLAUX);
```

- Fichero INLDNL_matlab.m:

```
% Script that obtains INL, DNL and missing codes

clear a DNL INL vtreal vtideal;
a=b(:,2);                           % a equals to column 2 of b
global misingcode;
global FS;
global INLCOD;
global DNLCOD;
global DNL;
global INL;

% Initializing variables
INL(:)=[];
DNL(:)=[];
INLCOD(:,:)=[];
DNLCOD(:,:)=[];
INL=[0];
DNL=[0];
INLCOD=[0];
DNLCOD=[0];
```

```
INLCOD(2,:)=[0];
DNLCOD(2,:)=[0];

tam2=size(a);
a(1:8)=0;
j=0;
aux=0;
monotonic=0;
misingcode=0;

% Evaluate if the output is monotonic and if it has got
% missing codes
for i=1:(tam2-1)
    if a(i)>a(i+1)
        aux=a(i);
        monotonic=monotonic+1;
    end
    if (a(i)<a(i+1))
        if (a(i)+1 == a(i+1) & a(i)>aux)
            j=j+1;
            vtreal(j)=i;
        else
            misingcode=misingcode+(a(i+1)-a(i)-1);
        end
    end
end

if(monotonic>0)                 % No monotonic output
    set(handles.Yes_Monotonic_Radiobutton,'Value',0.0);
    set(handles.No_Monotonic_Radiobutton,'Value',1.0);
else                            % Monotonic output
    set(handles.No_Monotonic_Radiobutton,'Value',0.0);
    set(handles.Yes_Monotonic_Radiobutton,'Value',1.0);
end

LSB=(vtreal(j)-vtreal(1))/(j-1);
vtideal(1)=vtreal(1);

% Evaluate INL and DNL
for i=1:j-1;
    DNL(i)=(vtreal(i+1)-vtreal(i)-LSB)/LSB;
    vtideal(i+1)=vtideal(i)+LSB;
    INL(i)=(vtreal(i)-vtideal(i))/LSB;
end
INL(j)=(vtreal(j)-vtideal(j))/LSB;

d=1;
for d=1:j-1;
     INLCOD(1,d)=(-FS/2)+(FS*((d-1)/(j-1)));
     INLCOD(2,d)=INL(d);
     DNLCOD(1,d)=(-FS/2)+(FS*(d/j));
     DNLCOD(2,d)=DNL(d);
end
INLCOD(1,j)=FS/2;
INLCOD(2,j)=INL(j);
```

- SimulacionNoIdeal.m:

```
% Function that controls single simulations
function SimulacionNoIdeal(Gain,Mismatch,SlewRate,TimeConstant,
     sigmajitter,sigma,Aux1,Aux2,Aux3,Aux4,handles,hObject)

global Ao;
global AodB;
global Eps;
global SR;
global Tau;
global INLMAX;
global DNLMAX;
global INL;
global DNL;
global misingcode;
global vector;
global vectorcode;
global INLCOD;
global DNLCOD;
global Select;
global Sndr;
global FS;
global bandwidth;
global minsignalband;
global modelsimulated;
global N;
global Ts;
global Voffset0;
global fsin;
global VoffsetVrefpn;
global nsamples;
global siginput;
global grphtitle;
global axisxlabelVolt;
global axisxlabelTime;
global axisxlabelFrequency;
global axisylabelINL;
global axisylabelDNL;
global axisylabelOutput;
global axisylabelInput;
global axisylabelSpectrum;
global Effort;
% Initializing variables
vector(:,:)=[];
vectorcode(:,:)=[];
vectorcode=[0];
vectorcode(2,:)=[0];
siginput(:,:)=[];

Eps=Mismatch;
SR=SlewRate;
AodB=Gain;
Tau=TimeConstant;
Ao = 10^(AodB/20);
axisxlabelVolt='Input voltages (V)';
axisxlabelTime='Time (Sec)';
axisxlabelFrequency='Frequency (Hz)';
axisylabelINL='INL';
```

```
axisylabelDNL='DNL';
axisylabelOutput='Output';
axisylabelInput='Input voltages (V)';
axisylabelSpectrum='Spectrum (dB)';

clear b;
set(handles.Messages_Listbox,'String','Simulation has started');

% Selection of simulation time depends on the input type
if (Select==1)                % Ramp Input
    sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
else                          % Sine Wave Input
    sim(modelsimulated,10*((Effort^2)+1)/fsin);
end
set(handles.Messages_Listbox,'String',char('Simulation has started',
    'Simulation has finished'));
vector=b;
tamvect=size(vector)
for l=1:tamvect(1),
    vectorcode(l,1)=(-FS/2)+(FS*vector(l,1)/(4*Effort*(2^N)*Ts*(1-
((2^(-N))/4))));
    vectorcode(l,2)=vector(l,2);
end

if (Select==1)                  % If input is ramp, calls a function
                                % that evaluate INL, DNL and
                                % missing codes
    CalcINLDNL(b,handles);
else                            % If input is sine wave, calls a
                                % function that evaluate SNDR,
                                % SNR and ENOB
    CalcSNDR(vector,bandwidth,minsignalband,nsamples,handles,hObject);
end
```

- Fichero EstudioGananciaParamGlobal.m:

```
% Function that implements Gain sweep

function EstudioGananciaParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global Ao;
global AodB;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global VoffsetVrefpn;
global Voffset0;
global INLMAX;
global vectsndr;
global N;
global Ts;
global fsin;
```

```
global Effort;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Gain sweep has started');

% Gain sweep from Start to Stop with Points points
for AodB=Start:(Stop-Start)/(Points-1):Stop
    Ao=10^(AodB/20)
    AodB
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
       sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
       sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain     sweep     has
started','Gain sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=AodB;
        INLMAX(2,k)=max(INLAUX)+misingcode;
        k=k+1;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr(1,k)=AodB;
        vectsndr(2,k)=Sndr;
        k=k+1;
    end
end
set(handles.Messages_Listbox,'String',char('Gain sweep has started',
      'Gain sweep is going on','Gain sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Finite Gain Sweep',
                  'Visible','on',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end
```

```
% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Finite Gain Sweep')
    xlabel('Opamp Gain (dB)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Finite Gain Sweep')
    xlabel('Opamp Gain (dB)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

AodB=str2double(get(handles.Gain,'String'))
Ao = 10^(AodB/20);
```

- Fichero EstudioMismatchEtapa1ParamGlobal.m:

```
% Function that implements Mismatch sweep

function EstudioMismatchEtapa1ParamGlobal(Start,Stop,
                Points,handles,hObject)

k=1;
global Eps;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global Ts;
global fsin;
global Effort;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Mismatch sweep has started');

% Mismatch sweep from Start to Stop with Points points
for Eps=Start:(Stop-Start)/(Points-1):Stop
    Eps
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;

    set(handles.Messages_Listbox,'String',char('Mismatch   sweep   has
started','Mismatch sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=Eps;
        INLMAX(2,k)=max(INLAUX)+misingcode;
        k=k+1;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
              nsamples,handles,hObject);
        vectsndr(1,k)=Eps;
        vectsndr(2,k)=Sndr;
        k=k+1;
    end
end

set(handles.Messages_Listbox,'String',char('Mismatch     sweep     has
started','Mismatch sweep is going on','Mismatch sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Mismatch Sweep',
                    'Visible','on',...
                    'NumberTitle','off','HandleVisibility','on',
                    'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Mismatch Sweep')
    xlabel('Capacitor Mismatch (%)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Mismatch Sweep')
    xlabel('Capacitor Mismatch (%)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

Eps = str2double(get(handles.Mismatch,'String'))
```

- Fichero EstudioSREtapa1ParamGlobal.m:

```
% Function that implements Slew Rate sweep

function EstudioSREtapa1ParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global SR;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global INLMAX;
global DNLMAX;
global vectsndr;
global Effort;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Slew Rate sweep has started');

% Slew Rate sweep from Start to Stop with Points points
for SR=Start:(Stop-Start)/(Points-1):Stop;
    SR
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Slew  Rate  sweep  has
started','Slew Rate sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=SR;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr(1,k)=SR;
```

```
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Slew Rate sweep has
started','Slew Rate sweep is going on','Slew Rate sweep has
finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Slew Rate Sweep',
                           'Visible','on',...
                          'NumberTitle','off','HandleVisibility','on',
                           'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Slew Rate Sweep')
    xlabel('Slew Rate')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Slew Rate Sweep')
    xlabel('Slew Rate')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end
SR=str2double(get(handles.SR,'String'))
```

- Fichero EstudioTauEtapa1ParamGlobal.m:

```
% Function that implements Tau sweep

function EstudioTauEtapa1ParamGlobal(Start,Stop,
               Points,handles,hObject)

k=1;
global Tau;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global INLMAX;
```

```
global DNLMAX;
global vectsndr;
global Effort;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Tau sweep has started');

% Tau sweep from Start to Stop with Points points
for Tau=Start:(Stop-Start)/(Points-1):Stop;
    Tau
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)          % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                    % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Tau      sweep      has
started','Tau sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=1/(2*pi*Tau);
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr(1,k)=1/(2*pi*Tau);
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Tau sweep has started',
      'Tau sweep is going on','Tau sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Tau Sweep',
                        'Visible','on',...
                        'NumberTitle','off','HandleVisibility','on',
                        'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end
```

```
% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Tau Sweep')
    xlabel('1/(2*pi*Tau)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Tau Sweep')
    xlabel('1/(2*pi*Tau)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end
Tau=str2double(get(handles.Tau,'String'))
```

- Fichero EstudioSigmaParamGlobal.m:

```
% Function that implements Sigma (offset) sweep

function EstudioSigmaParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global sigma;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global VoffsetVrefpn;
global Voffset0;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global Ts;
global fsin;
global Effort;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Sigma sweep has started');


% Sigma (offset) sweep from Start to Stop with Points points
for sigma=Start:(Stop-Start)/(Points-1):Stop
    sigma
    VoffsetVrefpn=sigma*randn(1);
    Voffset0=sigma*randn(1);
```

```
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end
    vector=b;
    set(handles.Messages_Listbox,'String',char('Sigma     sweep     has
started','Sigma sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=sigma;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
              nsamples,handles,hObject);
        vectsndr(1,k)=sigma;
        vectsndr(2,k)=Sndr;
    end
     k=k+1;
end

set(handles.Messages_Listbox,'String',char('Sigma sweep has started',
         'Sigma sweep is going on', 'Sigma sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Sigma Sweep',
                          'Visible','on',...
                          'NumberTitle','off','HandleVisibility','on',
                          'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Sigma Sweep')
    xlabel('Offset typical deviation')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Sigma Sweep')
    xlabel('Offset typical deviation')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end
sigma=str2double(get(handles.Sigma,'String'))
VoffsetVrefpn=sigma*randn(1);
Voffset0=sigma*randn(1);
```

- Fichero EstudioJitterParamGlobal.m:

```
% Function that implements Jitter sweep

function EstudioJitterParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global sigmajitter;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global Ts;
global fsin;
global Effort;
% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Jitter sweep has started');

% Jitter sweep from Start to Stop with Points points
for sigmajitter=Start:(Stop-Start)/(Points-1):Stop
    sigmajitter
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Jitter    sweep    has
started','Jitter sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=sigmajitter;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr(1,k)=sigmajitter;
```

```
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Jitter sweep has started',
      'Jitter sweep is going on','Jitter sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Jitter Sweep',
                           'Visible','on',...
                           'NumberTitle','off','HandleVisibility','on',
                           'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Jitter Sweep')
    xlabel('Jitter')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Jitter Sweep')
    xlabel('Jitter (sec)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

sigmajitter=str2double(get(handles.Jitter,'String'))
```

- Fichero EstudioSamplingFreqParamGlobal.m:

```
% Function that implements Sampling Frequency sweep

function EstudioSamplingFreqParamGlobal(Start,Stop,
                      Points,handles,hObject)

k=1;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global Ts;
```

```
global fsin;
global nsamples;
global Effort;
% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Sampling Frequency sweep has
started');

% Implement a Sampling Frequency sweep equals to implement a
% Sampling Time sweep where Sampling Time = 1/Sampling Frequency
for Ts=(1/Start):((1/Stop)-(1/Start))/(Points-1):(1/Stop)
    Ts
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Sampling      Frequency
sweep has started','Sampling Frequency sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=1/Ts;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr(1,k)=1/Ts;
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Sampling Frequency sweep
has started','Sampling Frequency sweep is going on','Sampling
Frequency sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Sampling Frequency Sweep',
                        'Visible','on',...
                        'NumberTitle','off','HandleVisibility','on',
                        'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end
```

```
% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Sampling Frequency Sweep')
    xlabel('Sampling Frequency (Hz)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Sampling Frequency Sweep')
    xlabel('Sampling Frequency (Hz)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

Ts=1/(str2double(get(handles.SamplingFrequency,'String')))
```

- Fichero EstudioAmplitParamGlobal.m:

```
% Function that implements input Amplitude sweep

function EstudioAmplitParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global A;
global Ts;
global FS;
global fsin;
global nsamples;
global Effort;
% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Amplitude sweep has started');

% Input Amplitude sweep from Start to Stop with Points points
for A=Start:(Stop-Start)/(Points-1):Stop
    A
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)                  % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                            % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Amplitude   sweep   has
started','Amplitude sweep is going on'));

    if (Select==1)                  % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=20*log(2*A/FS);
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                            % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr(1,k)=20*log(2*A/FS);
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Amplitude     sweep     has
started','Amplitude   sweep   is   going   on','Amplitude   sweep   has
finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Amplitude Sweep',
                  'Visible','on',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Amplitude Sweep')
    xlabel('Amplitude (dB)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Amplitude Sweep')
    xlabel('Amplitude (dB)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

A=str2double(get(handles.Amplitude,'String'))
```

- Fichero EstudioFSParamGlobal.m:

```
% Function that implements Full Scale sweep

function EstudioFSParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global FS;
global fsin;
global nsamples;
global Effort;
global Ts;
% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Full Scale sweep has started');

% Full Scale sweep from Start to Stop with Points points
for FS=Start:(Stop-Start)/(Points-1):Stop
    FS
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Full  Scale  sweep  has
started','Full Scale sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=FS;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
              nsamples,handles,hObject);
        vectsndr(1,k)=FS;
```

```
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Full Scale sweep has
started','Full Scale sweep is going on','Full Scale sweep has
finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Full Scale Sweep',
                          'Visible','on',...
                          'NumberTitle','off','HandleVisibility','on',
                          'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Full Scale Sweep')
    xlabel('Full Scale (Volts)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Full Scale Sweep')
    xlabel('Full Scale (Volts)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end
FS=str2double(get(handles.FullScale,'String'))
```

- Fichero EstudioFreqParamGlobal.m:

```
% Function that implements input Frequency sweep

function EstudioFreqParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global fsin;
global nsamples;
```

```
global Effort;
global Ts;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Frequency sweep has started');

% Input Frequency sweep from Start to Stop with Points points
for fsin=Start:(Stop-Start)/(Points-1):Stop
    fsin
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
       sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
       sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Frequency   sweep   has
started','Frequency sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=fsin;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr(1,k)=fsin;
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Frequency     sweep     has
started','Frequency   sweep   is   going   on','Frequency   sweep   has
finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Frequency Sweep',
                        'Visible','on',...
                        'NumberTitle','off','HandleVisibility','on',
                        'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end
```

```
% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Frequency Sweep')
    xlabel('Frequency (Hz)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Frequency Sweep')
    xlabel('Frequency (Hz)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end
fsin=str2double(get(handles.Frequency,'String'))
```

- Fichero EstudioBWParamGlobal.m:

```
% Function that implements Bandwidth sweep

function EstudioBWParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global fsin;
global nsamples;
global Effort;
global Ts;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Bandwidth sweep has started');

% Bandwidth sweep from Start to Stop with Points points
for bandwidth=Start:(Stop-Start)/(Points-1):Stop
    bandwidth
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Bandwidth   sweep   has
started','Bandwidth sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=bandwidth;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr(1,k)=bandwidth;
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Bandwidth     sweep     has
started','Bandwidth   sweep   is   going   on','Bandwidth   sweep   has
finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Bandwidth Sweep',
                    'Visible','on',...
                    'NumberTitle','off','HandleVisibility','on',
                    'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Bandwidth Sweep')
    xlabel('Bandwidth (Hz)')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Bandwidth Sweep')
    xlabel('Bandwidth (Hz)')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

bandwidth=str2double(get(handles.Bandwidth,'String'))
```

- Fichero EstudioAux1ParamGlobal.m:

```
% Function that implements Aux1 sweep

function EstudioAux1ParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global Ts;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global fsin;
global nsamples;
global Aux1;
global Effort;
% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Aux1 sweep has started');

% Aux1 sweep from Start to Stop with Points points
for Aux1=Start:(Stop-Start)/(Points-1):Stop
    Aux1
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)                  % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                            % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux1     sweep     has
started','Aux1 sweep is going on'));

    if (Select==1)                  % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=Aux1;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                            % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr(1,k)=Aux1;
```

```
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Aux1 sweep has started',
      'Aux1 sweep is going on','Aux1 sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux1 Sweep',
                        'Visible','on',...
                        'NumberTitle','off','HandleVisibility','on',
                        'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux1 Sweep')
    xlabel('Aux1')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux1 Sweep')
    xlabel('Aux1')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end
Aux1=str2double(get(handles.Aux1,'String'))
```

- Fichero EstudioAux2ParamGlobal.m:

```
% Function that implements Aux2 sweep

function EstudioAux2ParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global Ts;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global fsin;
global nsamples;
```

```
global Aux2;
global Effort;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Aux2 sweep has started');

% Aux2 sweep from Start to Stop with Points points
for Aux2=Start:(Stop-Start)/(Points-1):Stop
    Aux2
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)                          % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                                    % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux2     sweep     has
started','Aux2 sweep is going on'));

    if (Select==1)                          % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=Aux2;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr(1,k)=Aux2;
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Aux2 sweep has started',
      'Aux2 sweep is going on','Aux2 sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux2 Sweep',
                  'Visible','on',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end
```

```
% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux2 Sweep')
    xlabel('Aux2')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux2 Sweep')
    xlabel('Aux2')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

Aux2=str2double(get(handles.Aux2,'String'))
```

- Fichero EstudioAux3ParamGlobal.m:

```
% Function that implements Aux3 sweep

function EstudioAux3ParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global Ts;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global fsin;
global nsamples;
global Aux3;
global Effort;

% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Aux3 sweep has started');

% Aux3 sweep from Start to Stop with Points points
for Aux3=Start:(Stop-Start)/(Points-1):Stop
    Aux3
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux3     sweep     has
started','Aux3 sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=Aux3;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr(1,k)=Aux3;
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Aux3 sweep has started',
      'Aux3 sweep is going on','Aux3 sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux3 Sweep',
                  'Visible','on',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux3 Sweep')
    xlabel('Aux3')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux3 Sweep')
    xlabel('Aux3')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end

Aux3=str2double(get(handles.Aux3,'String'))
```

- Fichero EstudioAux4ParamGlobal.m:

```
% Function that implements Aux4 sweep

function EstudioAux4ParamGlobal(Start,Stop,Points,handles,hObject)

k=1;
global Ts;
global misingcode;
global bandwidth;
global minsignalband;
global nsamples;
global Select;
global Sndr;
global modelsimulated;
global INLMAX;
global DNLMAX;
global vectsndr;
global N;
global fsin;
global nsamples;
global Aux4;
global Effort;
% Initialize variables
INLMAX(:,:)=[];
INLMAX=[0];
INLMAX(2,:)=[0];
vectsndr(:,:)=[];
vectsndr=[0];
vectsndr(2,:)=[0];

set(handles.Messages_Listbox,'String','Aux4 sweep has started');

% Aux4 sweep from Start to Stop with Points points
for Aux4=Start:(Stop-Start)/(Points-1):Stop
    Aux4
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)          % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                    % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux4     sweep     has
started','Aux4 sweep is going on'));

    if (Select==1)          % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        INLMAX(1,k)=Aux4;
        INLMAX(2,k)=max(INLAUX)+misingcode;
    else                    % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr(1,k)=Aux4;
```

```
        vectsndr(2,k)=Sndr;
    end
    k=k+1;
end

set(handles.Messages_Listbox,'String',char('Aux4 sweep has started',
      'Aux4 sweep is going on','Aux4 sweep has finished'));

% If necessary, create the plot figure and store in handles structure
if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux4 Sweep',
                  'Visible','on',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
end

% Plot data
if (Select==1)
    plot(INLMAX(1,:), INLMAX(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux4 Sweep')
    xlabel('Aux4')
    ylabel('Max INL')
    grid(handles.PlotAxes,'on')
else
    plot(vectsndr(1,:), vectsndr(2,:),'Parent',handles.PlotAxes)
    set(0,'CurrentFigure',handles.PlotFigure)
    set(handles.PlotFigure,'Name','Aux4 Sweep')
    xlabel('Aux4')
    ylabel('Sndr (dB)')
    grid(handles.PlotAxes,'on')
end
Aux4=str2double(get(handles.Aux4,'String'))
```

- Fichero EstudioGananciaMismatchParamGlobal.m:

```
% Function that implements Gain and Mismatch sweep

function EstudioGananciaMismatchParamGlobal(Start,Stop,Points,
               Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global AodB;
global Eps;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vectSR;
global vectTau;
```

```
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Gain  and  Mismatch  sweep  has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Gain and Mismatch sweep',
                     'Visible','off',...
                     'NumberTitle','off','HandleVisibility','on',
                     'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Mismatch sweep from Start2 to Stop2 with Points2
% points inside Gain sweep from Start to Stop with Points points
for AodB=Start:(Stop-Start)/(Points-1):Stop
for Eps=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Ao=10^(AodB/20)
    AodB
    Eps
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain    and    Mismatch
sweep has started','Gain and Mismatch sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=Eps;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=AodB;
```

```
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
               nsamples,handles,hObject);
        vectsndr3d(k,g,1)=Eps;
        vectsndr3d(k,g,3)=AodB;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Gain and Mismatch sweep
has started','Gain and Mismatch sweep is going on','Gain and Mismatch
sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Gain and Mismatch sweep')

% Label axis and grid on
xlabel('Gain (dB)')
ylabel('Mismatch (%)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
AodB=str2double(get(handles.Gain,'String'))
Eps=str2double(get(handles.Mismatch,'String'))
Ao = 10^(AodB/20);
```

- Fichero EstudioGananciaSigmaParamGlobal.m:

```
% Function that implements Gain and Sigma (offset) sweep

function EstudioGananciaSigmaParamGlobal(Start,Stop,Points,
      Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global AodB;
global sigma;
global VoffsetVrefpn;
global Voffset0;
global modelsimulated;
global Select;
```

```
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Gain and Sigma sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Gain and Sigma sweep',
                     'Visible','off',...
                     'NumberTitle','off','HandleVisibility','on',
                     'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Sigma (offset) sweep from Start2 to Stop2 with Points2
% points inside Gain sweep from Start to Stop with Points points
for AodB=Start:(Stop-Start)/(Points-1):Stop
for sigma=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Ao=10^(AodB/20)
    AodB
    sigma
    VoffsetVrefpn = sigma * randn(1);
    Voffset0 = sigma * randn(1);
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end
    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain and Sigma sweep
has started','Gain and Sigma sweep is going on'));
```

```
    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=sigma;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=AodB;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigma;
        vectsndr3d(k,g,3)=AodB;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Gain and Sigma sweep has
started','Gain and Sigma sweep is going on','Gain and Sigma sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Gain and Sigma sweep')

% Label axis and grid on
xlabel('Gain (dB)')
ylabel('Offset typical deviation')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

AodB=str2double(get(handles.Gain,'String'))
sigma=str2double(get(handles.Sigma,'String'))
Ao = 10^(AodB/20);
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);
```

- Fichero EstudioGananciaJitterParamGlobal.m:

```
% Function that implements Gain and Jitter sweep

function EstudioGananciaJitterParamGlobal(Start,Stop,Points,
                Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global AodB;
global sigmajitter;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Gain   and   Jitter   sweep   has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Gain and Jitter sweep',
                    'Visible','off',...
                    'NumberTitle','off','HandleVisibility','on',
                    'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Jitter sweep from Start2 to Stop2 with Points2 points
% inside Gain sweep from Start to Stop with Points points
for AodB=Start:(Stop-Start)/(Points-1):Stop
for sigmajitter=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Ao=10^(AodB/20)
    AodB
    sigmajitter
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)                  % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                            % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end
    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain and Jitter sweep
has started','Gain and Jitter sweep is going on'));

    if (Select==1)                  % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=sigmajitter;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=AodB;
    else                           % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                  nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigmajitter;
        vectsndr3d(k,g,3)=AodB;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Gain and Jitter sweep has
started','Gain and Jitter sweep is going on','Gain and Jitter sweep
has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end

set(handles.PlotFigure,'Name','Gain and Jitter sweep')

% Label axis and grid on
xlabel('Gain (dB)')
ylabel('Jitter (sec)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

AodB=str2double(get(handles.Gain,'String'))
sigmajitter=str2double(get(handles.Jitter,'String'))
Ao = 10^(AodB/20);
```

- Fichero EstudioGananciaSamplingFreqParamGlobal.m:

```
% Function that implements Gain and Sampling Frequency sweep

function EstudioGananciaSamplingFreqParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global AodB;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Gain  and  Sampling  Frequency
sweep has started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Gain and Sampling Frequency
sweep','Visible','off',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Implement a Sampling Frequency sweep equals to implement a
% Sampling Time sweep where Sampling Time = 1/Sampling Frequency
% Double sweep: Sampling Time sweep from 1/Start2 to 1/Stop2 with
% Points2 points inside Gain sweep from Start to Stop with Points
points
for AodB=Start:(Stop-Start)/(Points-1):Stop
for Ts=(1/Start2):((1/Stop2)-(1/Start2))/(Points2-1):(1/Stop2);
    Ao=10^(AodB/20)
    AodB
    Ts
```

```
    clear b;
    clear vector;
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
       sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
       sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end
    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain    and    Sampling
Frequency  sweep  has  started','Gain  and  Sampling  Frequency  sweep  is
going on'));
    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=1/Ts;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=AodB;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=1/Ts;
        vectsndr3d(k,g,3)=AodB;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Gain      and      Sampling
Frequency  sweep  has  started','Gain  and  Sampling  Frequency  sweep  is
going on','Gain and Sampling Frequency sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Gain and Sampling Frequency sweep')

% Label axis and grid on
xlabel('Gain (dB)')
ylabel('Sampling Frequency (Hz)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
AodB=str2double(get(handles.Gain,'String'))
Ts=1/(str2double(get(handles.SamplingFrequency,'String')))
Ao = 10^(AodB/20);
```

- Fichero EstudioGananciaAmplitParamGlobal.m:

```
% Function that implements Gain and input Amplitude sweep

function EstudioGananciaAmplitParamGlobal(Start,Stop,Points,
      Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;

global AodB;
global A;
global modelsimulated;
global Select;
global N;
global Ts;
global FS;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Gain  and  Amplitude  sweep  has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure    =    figure('Name','Gain    and    Amplitude
sweep','Visible','off',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end
```

```
% Double sweep: Input Amplitude sweep from Start2 to Stop2 with
% Points2 points inside Gain sweep from Start to Stop with
% Points points
for AodB=Start:(Stop-Start)/(Points-1):Stop
for A=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Ao=10^(AodB/20)
    AodB
    A
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain   and   Amplitude
sweep has started','Gain and Amplitude sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=20*log(2*A/FS);
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=AodB;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=20*log(2*A/FS);
        vectsndr3d(k,g,3)=AodB;
        vectsndr3d(k,g,2)=Sndr;
    end
    k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Gain and Amplitude sweep
has started','Gain and Amplitude sweep is going on','Gain and
Amplitude sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Gain and Amplitude sweep')
```

```
% Label axis and grid on
xlabel('Gain (dB)')
ylabel('Amplitude (dB)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

AodB=str2double(get(handles.Gain,'String'))
A=str2double(get(handles.Amplitude,'String'))
Ao = 10^(AodB/20);
```

- Fichero EstudioGananciaFSParamGlobal.m:

```
% Function that implements Gain and Full Scale sweep

function EstudioGananciaFSParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;

global AodB;
global FS;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Gain and Full Scale sweep has
started');
```

```
% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Gain and Full Scale sweep',
                  'Visible','off',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Full Scale sweep from Start2 to Stop2 with
% Points2 points inside Gain sweep from Start to Stop with
% Points points
for AodB=Start:(Stop-Start)/(Points-1):Stop
for FS=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Ao=10^(AodB/20)
    AodB
    FS
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
       sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
       sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain  and  Full  Scale
sweep has started','Gain and Full Scale sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=FS;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=AodB;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=FS;
        vectsndr3d(k,g,3)=AodB;
        vectsndr3d(k,g,2)=Sndr;
    end
    k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Gain and Full Scale sweep
has started','Gain and Full Scale sweep is going on','Gain and Full
Scale sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)
```

```
% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Gain and Full Scale sweep')

% Label axis and grid on
xlabel('Gain (dB)')
ylabel('Full Scale (Volts)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
AodB=str2double(get(handles.Gain,'String'))
FS=str2double(get(handles.FullScale,'String'))
Ao = 10^(AodB/20);
```

- Fichero EstudioGananciaFreqParamGlobal.m:

```
% Function that implements Gain and input Frequency sweep

function EstudioGananciaFreqParamGlobal(Start,Stop,Points,
                 Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global AodB;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
```

```
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Gain and Frequency sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Gain and Frequency sweep',
                'Visible','off',...
                'NumberTitle','off','HandleVisibility','on',
                'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Input Frequency sweep from Start2 to Stop2 with
% Points2 points inside Gain sweep from Start to Stop with
% Points points
for AodB=Start:(Stop-Start)/(Points-1):Stop
for fsin=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Ao=10^(AodB/20)
    AodB
    fsin
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Gain   and   Frequency
sweep has started','Gain and Frequency sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=fsin;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=AodB;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=fsin;
        vectsndr3d(k,g,3)=AodB;
        vectsndr3d(k,g,2)=Sndr;
    end
    k=k+1;
end
g=g+1;
k=1;
end
set(handles.Messages_Listbox,'String',char('Gain and Frequency sweep
has started','Gain and Frequency sweep is going on','Gain and
Frequency sweep has finished'));
```

```
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Gain and Frequency sweep')

% Label axis and grid on
xlabel('Gain (dB)')
ylabel('Frequency (Hz)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
AodB=str2double(get(handles.Gain,'String'))
fsin=str2double(get(handles.Frequency,'String'))
Ao = 10^(AodB/20);
```

- Fichero EstudioMismatchSigmaParamGlobal.m:

```
% Function that implements Mismatch and Sigma (offset) sweep

function EstudioMismatchSigmaParamGlobal(Start,Stop,Points,
               Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Eps;
global sigma;
global VoffsetVrefpn;
global Voffset0;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
```

```
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Mismatch and Sigma sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Mismatch and Sigma sweep',
                'Visible','off',...
                'NumberTitle','off','HandleVisibility','on',
                'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Sigma (offset) sweep from Start2 to Stop2 with
% Points2 points inside Mismatch sweep from Start to Stop with
% Points points
for Eps=Start:(Stop-Start)/(Points-1):Stop
for sigma=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Eps
    sigma
    VoffsetVrefpn = sigma * randn(1);
    Voffset0 = sigma * randn(1);
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end
    vector=b;
    set(handles.Messages_Listbox,'String',char('Mismatch   and   Sigma
sweep has started','Mismatch and Sigma sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        DNLAUX=abs(DNL);
        maxINL(k,g,1)=sigma;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Eps;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigma;
        vectsndr3d(k,g,3)=Eps;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
```

```
k=1;
end
set(handles.Messages_Listbox,'String',char('Mismatch and Sigma sweep
has started','Mismatch and Sigma sweep is going on','Mismatch and
Sigma sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Mismatch and Sigma sweep')

% Label axis and grid on
xlabel('Mismatch (%)')
ylabel('Offset typical deviation')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
Eps=str2double(get(handles.Mismatch,'String'))
sigma=str2double(get(handles.Sigma,'String'))
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);
```

- Fichero EstudioMismatchJitterParamGlobal.m:

```
% Function that implements Mismatch and Jitter sweep

function EstudioMismatchJitterParamGlobal(Start,Stop,Points,
               Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Eps;
global sigmajitter;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
```

```
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Mismatch and Jitter sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Mismatch and Jitter sweep',
                'Visible','off',...
                'NumberTitle','off','HandleVisibility','on',
                'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Jitter sweep from Start2 to Stop2 with Points2 points
% inside Mismatch sweep from Start to Stop with Points points
for Eps=Start:(Stop-Start)/(Points-1):Stop
for sigmajitter=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Eps
    sigmajitter
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end
    vector=b;
    set(handles.Messages_Listbox,'String',char('Mismatch   and   Jitter
sweep has started','Mismatch and Jitter sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=sigmajitter;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Eps;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigmajitter;
        vectsndr3d(k,g,3)=Eps;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
```

```
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Mismatch and Jitter sweep
has started','Mismatch and Jitter sweep is going on','Mismatch and
Jitter sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Mismatch and Jitter sweep')

% Label axis and grid on
xlabel('Mismatch (%)')
ylabel('Jitter (sec)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
Eps=str2double(get(handles.Mismatch,'String'))
sigmajitter=str2double(get(handles.Jitter,'String'))
```

- Fichero EstudioMismatchSamplingFreqParamGlobal.m:

```
% Function that implements Mismatch and Sampling Frequency sweep

function EstudioMismatchSamplingFreqParamGlobal(Start,Stop,Points,
             Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Eps;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
```

```
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Mismatch and Sampling Frequency
sweep has started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure   =   figure('Name','Mismatch   and   Sampling
Frequency sweep','Visible','off',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Implement a Sampling Frequency sweep equals to implement a
% Sampling Time sweep where Sampling Time = 1/Sampling Frequency
% Double sweep: Sampling Time sweep from 1/Start2 to 1/Stop2 with
% Points2 points inside Mismatch sweep from Start to Stop with
% Points points
for Eps=Start:(Stop-Start)/(Points-1):Stop
for Ts=(1/Start2):((1/Stop2)-(1/Start2))/(Points2-1):(1/Stop2);
    Eps
    Ts
    clear b;
    clear vector;

    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end
    vector=b;
    set(handles.Messages_Listbox,'String',char('Mismatch  and  Sampling
Frequency sweep has started','Mismatch and Sampling Frequency sweep is
going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=1/Ts;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Eps;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
```

```
        vectsndr3d(k,g,1)=1/Ts;
        vectsndr3d(k,g,3)=Eps;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Mismatch and Sampling
Frequency sweep has started','Mismatch and Sampling Frequency sweep is
going on','Mismatch and Sampling Frequency sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Mismatch and Sampling Frequency sweep')

% Label axis and grid on
xlabel('Mismatch (%)')
ylabel('Sampling Frequency (Hz)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
Eps=str2double(get(handles.Mismatch,'String'))
Ts=1/(str2double(get(handles.SamplingFrequency,'String')))
```

- Fichero EstudioMismatchAmplitParamGlobal.m:

```
% Function that implements Mismatch and input Amplitude sweep

function EstudioMismatchAmplitParamGlobal(Start,Stop,Points,
                    Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Eps;
global A;
global modelsimulated;
global Select;
global N;
global Ts;
global FS;
global fsin;
global maxINL;
global DNLMAX;
```

```
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Mismatch and Amplitude sweep
has started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Mismatch and Amplitude
sweep','Visible','off',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Input Amplitude sweep from Start2 to Stop2 with
% Points2 points inside Mismatch sweep from Start to Stop with
% Points points
for Eps=Start:(Stop-Start)/(Points-1):Stop
for A=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Eps
    A
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Mismatch and Amplitude
sweep has started','Mismatch and Amplitude sweep is going on'));
```

```
    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=20*log(2*A/FS);
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Eps;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=20*log(2*A/FS);
        vectsndr3d(k,g,3)=Eps;
        vectsndr3d(k,g,2)=Sndr;
    end
    k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Mismatch and Amplitude
sweep has started','Mismatch and Amplitude sweep is going
on','Mismatch and Amplitude sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Mismatch and Amplitude sweep')

% Label axis and plot
xlabel('Mismatch (%)')
ylabel('Amplitude (dB)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Eps=str2double(get(handles.Mismatch,'String'))
A=str2double(get(handles.Amplitude,'String'))
```

- Fichero EstudioMismatchFSParamGlobal.m:

```
% Function that implements Mismatch and Full Scale sweep

function EstudioMismatchFSParamGlobal(Start,Stop,Points,
                  Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Eps;
global FS;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Mismatch and Full Scale sweep
has started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Mismatch and Full Scale
sweep','Visible','off',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Full Scale sweep from Start2 to Stop2 with
% Points2 points inside Mismatch sweep from Start to Stop with
% Points points
for Eps=Start:(Stop-Start)/(Points-1):Stop
for FS=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Eps
    FS
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Mismatch    and    Full
Scale  sweep  has  started','Mismatch  and  Full  Scale  sweep  is  going
on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=FS;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Eps;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=FS;
        vectsndr3d(k,g,3)=Eps;
        vectsndr3d(k,g,2)=Sndr;
    end
    k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Mismatch  and  Full  Scale
sweep  has  started','Mismatch  and  Full  Scale  sweep  is  going
on','Mismatch and Full Scale sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Mismatch and Full Scale sweep')

% Label axis and grid on
xlabel('Mismatch (%)')
ylabel('Full Scale (Volts)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Eps=str2double(get(handles.Mismatch,'String'))
FS=str2double(get(handles.FullScale,'String'))
```

- Fichero EstudioMismatchFreqParamGlobal.m:

```
% Function that implements Mismatch and input Frequency sweep

function EstudioMismatchFreqParamGlobal(Start,Stop,Points,
             Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Eps;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Mismatch  and  Frequency  sweep
has started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure   =   figure('Name','Mismatch   and   Frequency
sweep','Visible','off',...
      'NumberTitle','off','HandleVisibility','on',
      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Input Frequency sweep from Start2 to Stop2 with
% Points2 points inside Mismatch sweep from Start to Stop with
% Points points
for Eps=Start:(Stop-Start)/(Points-1):Stop
for fsin=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Eps
    fsin
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Mismatch and Frequency
sweep has started','Mismatch and Frequency sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=fsin;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Eps;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=fsin;
        vectsndr3d(k,g,3)=Eps;
        vectsndr3d(k,g,2)=Sndr;
    end
    k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Mismatch   and   Frequency
sweep   has   started','Mismatch   and   Frequency   sweep   is   going
on','Mismatch and Frequency sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Mismatch and Frequency sweep')

% Label axis and grid on
xlabel('Mismatch (%)')
ylabel('Frequency (Hz)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Eps=str2double(get(handles.Mismatch,'String'))
fsin=str2double(get(handles.Frequency,'String'))
```

- Fichero EstudioSRTauEtapa1ParamGlobal.m:

```
% Function that implements Slew Rate and Tau sweep

function EstudioSRTauEtapa1ParamGlobal(Start,Stop,Points,
           Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global SR;
global Tau;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vectSR;
global vectTau;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Slew Rate and Tau sweep has
started');
% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Slew Rate and Tau sweep',
                   'Visible','off',...
                   'NumberTitle','off','HandleVisibility','on',
                   'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Tau sweep from Start2 to Stop2 with Points2 points
% inside Slew Rate sweep from Start to Stop with Points points
for SR=Start:(Stop-Start)/(Points-1):Stop;
for Tau=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    SR
    Tau
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Slew   Rate   and   Tau
sweep has started','Slew Rate and Tau sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=1/(Tau*2*pi);
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=SR;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=1/(Tau*2*pi);
        vectsndr3d(k,g,3)=SR;
        vectsndr3d(k,g,2)=Sndr;
    end
    k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Slew  Rate  and  Tau  sweep
has started','Slew Rate and Tau sweep is going on','Slew Rate and Tau
sweep has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Slew Rate and Tau sweep')

% Label axis and grid on
xlabel('Slew Rate')
ylabel('1/(Tau*2*pi)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

SR=str2double(get(handles.SR,'String'))
Tau=str2double(get(handles.Tau,'String'))
```

- Fichero EstudioSRSigmaParamGlobal.m:

```
% Function that implements Slew Rate and Sigma (offset) sweep

function EstudioSRSigmaParamGlobal(Start,Stop,Points,
              Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;

global SR;
global sigma;
global VoffsetVrefpn;
global Voffset0;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Slew Rate and Sigma sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Slew Rate and Sigma sweep',
              'Visible','off',...
              'NumberTitle','off','HandleVisibility','on',
              'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end
```

```
% Double sweep: Sigma (offset) sweep from Start2 to Stop2 with
% Points2 points inside Slew Rate sweep from Start to Stop with
% Points points
for SR=Start:(Stop-Start)/(Points-1):Stop
for sigma=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    SR
    sigma
    VoffsetVrefpn = sigma * randn(1);
    Voffset0 = sigma * randn(1);
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Slew  Rate  and  Sigma
sweep has started','Slew Rate and Sigma sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=sigma;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=SR;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigma;
        vectsndr3d(k,g,3)=SR;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Slew Rate and Sigma sweep
has started',
'Slew Rate and Sigma sweep is going on','Slew Rate and Sigma sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Slew Rate and Sigma sweep')
```

```
% Label axis and grid on
xlabel('Slew Rate')
ylabel('Offset typical deviation')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
SR=str2double(get(handles.SR,'String'))
sigma=str2double(get(handles.Sigma,'String'))
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);
```

- Fichero EstudioSRJitterParamGlobal.m:

```
% Function that implements Slew Rate and Jitter sweep

function EstudioSRJitterParamGlobal(Start,Stop,Points,
              Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global SR;
global sigmajitter;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Slew Rate and Jitter sweep has
started');
```

```
% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Slew Rate and Jitter sweep',
              'Visible','off',...
              'NumberTitle','off','HandleVisibility','on',
              'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Jitter sweep from Start2 to Stop2 with Points2 points
% inside Slew Rate sweep from Start to Stop with Points points
for SR=Start:(Stop-Start)/(Points-1):Stop
for sigmajitter=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    SR
    sigmajitter
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Slew  Rate  and  Jitter
sweep has started','Slew Rate and Jitter sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=sigmajitter;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=SR;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigmajitter;
        vectsndr3d(k,g,3)=SR;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Slew Rate and Jitter sweep
has started',
'Slew Rate and Jitter sweep is going on','Slew Rate and Jitter sweep
has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)
```

```
% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))

else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Slew Rate and Jitter sweep')

% Label axis and grid on
xlabel('Slew Rate')
ylabel('Jitter (sec)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

SR=str2double(get(handles.SR,'String'))
sigmajitter=str2double(get(handles.Jitter,'String'))
```

- Fichero EstudioTauSigmaParamGlobal.m:

```
% Function that implements Tau and Sigma (offset) sweep

function EstudioTauSigmaParamGlobal(Start,Stop,Points,
               Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;

global Tau;
global sigma;
global VoffsetVrefpn;
global Voffset0;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
```

```
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Tau and Sigma sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Tau and Sigma sweep',
              'Visible','off',...
              'NumberTitle','off','HandleVisibility','on',
              'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Sigma (offset) sweep from Start2 to Stop2 with
% Points2 points inside Tau sweep from Start to Stop with
% Points points
for Tau=Start:(Stop-Start)/(Points-1):Stop
for sigma=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Tau
    sigma
    VoffsetVrefpn = sigma * randn(1);
    Voffset0 = sigma * randn(1);
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Tau  and  Sigma  sweep
has started','Tau and Sigma sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        DNLAUX=abs(DNL);
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=1/(2*pi*Tau);
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigma;
        vectsndr3d(k,g,3)=1/(2*pi*Tau);
        vectsndr3d(k,g,2)=Sndr;
    end
```

```
     k=k+1;
end
g=g+1;
k=1;
end
set(handles.Messages_Listbox,'String',char('Tau and Sigma sweep has
started','Tau and Sigma sweep is going on','Tau and Sigma sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Tau and Sigma sweep')

% Label axis and grid on
xlabel('1/(2*pi*Tau)')
ylabel('Offset typical deviation')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
Tau=str2double(get(handles.Tau,'String'))
sigma=str2double(get(handles.Sigma,'String'))
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);
```

- Fichero EstudioTauJitterParamGlobal.m:

```
% Function that implements Tau and Jitter sweep

function EstudioTauJitterParamGlobal(Start,Stop,Points,
                  Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Tau;
global sigmajitter;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
```

```
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Tau and Jitter sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Tau and Jitter sweep',
                    'Visible','off',...
                    'NumberTitle','off','HandleVisibility','on',
                    'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Jitter sweep from Start2 to Stop2 with Points2 points
% inside Tau sweep from Start to Stop with Points points
for Tau=Start:(Stop-Start)/(Points-1):Stop
for sigmajitter=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Tau
    sigmajitter
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Tau  and  Jitter  sweep
has started','Tau and Jitter sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=sigmajitter;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=1/(2*pi*Tau);
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                  nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigmajitter;
        vectsndr3d(k,g,3)=1/(2*pi*Tau);
```

```
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end
set(handles.Messages_Listbox,'String',char('Tau and Jitter sweep has
started','Tau and Jitter sweep is going on','Tau and Jitter sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Tau and Jitter sweep')

% Label axis and grid on
xlabel('1/(2*pi*Tau)')
ylabel('Jitter (sec)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')
Tau=str2double(get(handles.Tau,'String'))
sigmajitter=str2double(get(handles.Jitter,'String'))
```

- Fichero EstudioSigmaJitterParamGlobal.m:

```
% Function that implements Sigma (offset) and Jitter sweep

function EstudioSigmaJitterParamGlobal(Start,Stop,Points,
                  Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global sigma;
global VoffsetVrefpn;
global Voffset0;
global sigmajitter;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
```

```
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Sigma and Jitter sweep has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Sigma and Jitter sweep',
                  'Visible','off',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Jitter sweep from Start2 to Stop2 with Points2 points
% inside Sigma (offset) sweep from Start to Stop with Points points
for sigma=Start:(Stop-Start)/(Points-1):Stop
for sigmajitter=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    sigma
    VoffsetVrefpn = sigma * randn(1);
    Voffset0 = sigma * randn(1);
    sigmajitter
    clear b;
    clear vector;

% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Sigma and Jitter sweep
has started','Sigma and Jitter sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=sigmajitter;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=sigma;
```

```
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=sigmajitter;
        vectsndr3d(k,g,3)=sigma;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Sigma and Jitter sweep has
started','Sigma and Jitter sweep is going on','Sigma and Jitter sweep
has finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Sigma and Jitter sweep')

% Label axis and grid on
xlabel('Offset typical deviation')
ylabel('Jitter (sec)')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

sigma=str2double(get(handles.Sigma,'String'))
VoffsetVrefpn = sigma * randn(1);
Voffset0 = sigma * randn(1);
sigmajitter=str2double(get(handles.Jitter,'String'))
```

- Fichero EstudioAux1Aux2ParamGlobal.m:

```
% Function that implements Aux1 and Aux2 sweep

function EstudioAux1Aux2ParamGlobal(Start,Stop,Points,
                      Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Aux1;
global Aux2;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];


set(handles.Messages_Listbox,'String','Aux1   and   Aux2   sweep   has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux1 and Aux2 sweep',
                'Visible','off',...
                'NumberTitle','off','HandleVisibility','on',
                'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Aux2 sweep from Start2 to Stop2 with Points2 points
% inside Aux1 sweep from Start to Stop with Points points
for Aux1=Start:(Stop-Start)/(Points-1):Stop
for Aux2=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Aux1
    Aux2
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux1  and  Aux2  sweep
has started','Aux1 and Aux2 sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=Aux2;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Aux1;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=Aux2;
        vectsndr3d(k,g,3)=Aux1;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Aux1  and  Aux2  sweep  has
started','Aux1 and Aux2 sweep is going on','Aux1 and Aux2 sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Aux1 and Aux2 sweep')

% Label axis and grid on
xlabel('Aux1')
ylabel('Aux2')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Aux1=str2double(get(handles.Aux1,'String'))
Aux2=str2double(get(handles.Aux2,'String'))
```

- Fichero EstudioAux1Aux3ParamGlobal.m:

```
% Function that implements Aux1 and Aux3 sweep

function EstudioAux1Aux3ParamGlobal(Start,Stop,Points,
                 Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Aux1;
global Aux3;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;

% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Aux1   and   Aux3   sweep   has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux1 and Aux3 sweep',
                  'Visible','off',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Aux3 sweep from Start2 to Stop2 with Points2 points
% inside Aux1 sweep from Start to Stop with Points points
for Aux1=Start:(Stop-Start)/(Points-1):Stop
for Aux3=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Aux1
    Aux3
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux1  and  Aux3  sweep
has started','Aux1 and Aux3 sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=Aux3;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Aux1;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=Aux3;
        vectsndr3d(k,g,3)=Aux1;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Aux1  and  Aux3  sweep  has
started','Aux1 and Aux3 sweep is going on','Aux1 and Aux3 sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Aux1 and Aux3 sweep')

% Label axis and grid on
xlabel('Aux1')
ylabel('Aux3')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Aux1=str2double(get(handles.Aux1,'String'))
Aux3=str2double(get(handles.Aux3,'String'))
```

- Fichero EstudioAux1Aux4ParamGlobal.m:

```
% Function that implements Aux1 and Aux4 sweep

function EstudioAux1Aux4ParamGlobal(Start,Stop,Points,
              Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Aux1;
global Aux4;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Aux1   and   Aux4   sweep   has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux1 and Aux4 sweep',
                  'Visible','off',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Aux4 sweep from Start2 to Stop2 with Points2 points
% inside Aux1 sweep from Start to Stop with Points points
for Aux1=Start:(Stop-Start)/(Points-1):Stop
for Aux4=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Aux1
    Aux4
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux1  and  Aux4  sweep
has started','Aux1 and Aux4 sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=Aux4;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Aux1;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                nsamples,handles,hObject);
        vectsndr3d(k,g,1)=Aux4;
        vectsndr3d(k,g,3)=Aux1;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Aux1  and  Aux4  sweep  has
started','Aux1 and Aux4 sweep is going on','Aux1 and Aux4 sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Aux1 and Aux4 sweep')

% Label axis and grid on
xlabel('Aux1')
ylabel('Aux4')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Aux1=str2double(get(handles.Aux1,'String'))
Aux4=str2double(get(handles.Aux4,'String'))
```

- Fichero EstudioAux2Aux3ParamGlobal.m:

```
% Function that implements Aux2 and Aux3 sweep

function EstudioAux2Aux3ParamGlobal(Start,Stop,Points,
                   Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Aux2;
global Aux3;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Aux2   and   Aux3   sweep   has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux2 and Aux3 sweep',
                'Visible','off',...
                'NumberTitle','off','HandleVisibility','on',
                'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Aux3 sweep from Start2 to Stop2 with Points2 points
% inside Aux2 sweep from Start to Stop with Points points
for Aux2=Start:(Stop-Start)/(Points-1):Stop
for Aux3=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Aux2
    Aux3
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux2  and  Aux3  sweep
has started','Aux2 and Aux3 sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=Aux3;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Aux2;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                  nsamples,handles,hObject);
        vectsndr3d(k,g,1)=Aux3;
        vectsndr3d(k,g,3)=Aux2;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Aux2  and  Aux3  sweep  has
started','Aux2 and Aux3 sweep is going on','Aux2 and Aux3 sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Aux2 and Aux3 sweep')

% Label axis and grid on
xlabel('Aux2')
ylabel('Aux3')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Aux2=str2double(get(handles.Aux2,'String'))
Aux3=str2double(get(handles.Aux3,'String'))
```

- Fichero EstudioAux2Aux4ParamGlobal.m:

```
% Function that implements Aux2 and Aux4 sweep

function EstudioAux2Aux4ParamGlobal(Start,Stop,Points,
                Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Aux2;
global Aux4;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Aux2   and   Aux4   sweep   has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux2 and Aux4 sweep',
                      'Visible','off',...
                      'NumberTitle','off','HandleVisibility','on',
                      'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Aux4 sweep from Start2 to Stop2 with Points2 points
% inside Aux2 sweep from Start to Stop with Points points
for Aux2=Start:(Stop-Start)/(Points-1):Stop
for Aux4=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Aux2
    Aux4
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
       sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
       sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux2  and  Aux4  sweep
has started','Aux2 and Aux4 sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=Aux4;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Aux2;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=Aux4;
        vectsndr3d(k,g,3)=Aux2;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Aux2  and  Aux4  sweep  has
started','Aux2 and Aux4 sweep is going on','Aux2 and Aux4 sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Aux2 and Aux4 sweep')

% Label axis and grid on
xlabel('Aux2')
ylabel('Aux4')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Aux2=str2double(get(handles.Aux2,'String'))
Aux4=str2double(get(handles.Aux4,'String'))
```

- Fichero EstudioAux3Aux4ParamGlobal.m:

```
% Function that implements Aux3 and Aux4 sweep

function EstudioAux3Aux4ParamGlobal(Start,Stop,Points,
               Start2,Stop2,Points2,handles,hObject)

k=1;
g=1;
global Aux3;
global Aux4;
global modelsimulated;
global Select;
global N;
global Ts;
global fsin;
global maxINL;
global DNLMAX;
global vectsndr3d;
global vector;
global bandwidth;
global minsignalband;
global nsamples;
global Sndr;
global S;
global T;
global Effort;
% Initialize variables
maxINL(:,:,:)=[];
maxINL=[0];
maxINL(2,:)=[0];
maxINL(3,:)=[0];
vectsndr3d(:,:,:)=[];
vectsndr3d=[0];
vectsndr3d(2,:)=[0];
vectsndr3d(3,:)=[0];

set(handles.Messages_Listbox,'String','Aux3   and   Aux4   sweep   has
started');

% If necessary, create the plot figure and store in handles structure
 if ~isfield(handles,'PlotFigure') | ~ishandle(handles.PlotFigure),
      handles.PlotFigure = figure('Name','Aux3 and Aux4 sweep',
                  'Visible','off',...
                  'NumberTitle','off','HandleVisibility','on',
                  'IntegerHandle','off');
      handles.PlotAxes = axes('Parent',handles.PlotFigure);
      guidata(hObject,handles)
 end

% Double sweep: Aux4 sweep from Start2 to Stop2 with Points2 points
% inside Aux3 sweep from Start to Stop with Points points
for Aux3=Start:(Stop-Start)/(Points-1):Stop
for Aux4=Start2:(Stop2-Start2)/(Points2-1):Stop2;
    Aux3
    Aux4
    clear b;
    clear vector;
```

```
% Selection of simulation time depends on the input type
    if (Select==1)              % Ramp input
        sim(modelsimulated,4*Effort*(2^N)*Ts*(1-((2^(-N))/4)));
    else                        % Sine wave input
        sim(modelsimulated,10*((Effort^2)+1)/fsin);
    end

    vector=b;
    set(handles.Messages_Listbox,'String',char('Aux3  and  Aux4  sweep
has started','Aux3 and Aux4 sweep is going on'));

    if (Select==1)              % Ramp input => Evaluate INL
        INLDNL_matlab
        INLAUX=abs(INL);
        maxINL(k,g,1)=Aux4;
        maxINL(k,g,2)=max(INLAUX)+misingcode;
        maxINL(k,g,3)=Aux3;
    else                        % Sine wave input => Evaluate SNDR
        CalcSNDR(vector,bandwidth,minsignalband,
                 nsamples,handles,hObject);
        vectsndr3d(k,g,1)=Aux4;
        vectsndr3d(k,g,3)=Aux3;
        vectsndr3d(k,g,2)=Sndr;
    end
     k=k+1;
end
g=g+1;
k=1;
end

set(handles.Messages_Listbox,'String',char('Aux3  and  Aux4  sweep  has
started','Aux3 and Aux4 sweep is going on','Aux3 and Aux4 sweep has
finished'));
set(handles.PlotFigure,'Visible','on')
set(0,'CurrentFigure',handles.PlotFigure)

% Prepare data and plot
if (Select==1)
    [S,T]=meshgrid(maxINL(1,:,3),maxINL(:,1,1));
    surf(S,T,maxINL(:,:,2))
else
    [S,T]=meshgrid(vectsndr3d(1,:,3),vectsndr3d(:,1,1));
    surf(S,T,vectsndr3d(:,:,2))
end
set(handles.PlotFigure,'Name','Aux3 and Aux4 sweep')

% Label axis and grid on
xlabel('Aux3')
ylabel('Aux4')
if(Select==1)
    zlabel('Max INL')
else
    zlabel('Sndr (dB)')
end
grid(handles.PlotAxes,'on')

Aux3=str2double(get(handles.Aux3,'String'))
Aux4=str2double(get(handles.Aux4,'String'))
```